АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Лекция 1

План

- Алгоритм. Свойства алгоритма. Способы описания алгоритма
- Блок-схемы алгоритмов
- Введение в Python
- Первая программа на Python
- Переменные и типы данных
- Операции с числами
- Условные выражения
- Циклы
- Функции

Алгоритм. Свойства алгоритма. Способы описания алгоритма

Если мы хотим написать программу на каком-либо языке программирования, то сначала нам следует составить алгоритм решения задачи.

Алгоритм - это точное и простое описание последовательности действий для решения данной задачи. Алгоритм содержит несколько шагов, которые должны выполняться в определенной последовательности. Каждый шаг алгоритма может состоять из одной или нескольких простых операций.

Алгоритм. Свойства алгоритма. Способы описания алгоритма

Каждый из нас ежедневно использует различные алгоритмы: инструкции, правила, рецепты и т. д. Обычно мы это делаем не задумываясь. Например, открывая дверь ключом, никто не размышляет над тем, в какой последовательности выполнять действия. Однако чтобы кого-нибудь научить открывать дверь, придется четко указать и сами действия, и порядок их выполнения. Например:

1. Достать ключ.
2. Вставить ключ в замочную скважину.
3. Повернуть ключ два раза против часовой стрелки.
4. Вынуть ключ.

Алгоритм. Свойства алгоритма. Способы описания алгоритма

Представим, что мы поменяли местами второе и третье действия. Мы сможем выполнить и этот алгоритм, но дверь не откроется, т. е. алгоритм станет невыполнимым.

Для алгоритма важен не только набор действий, но и то, в каком порядке они выполняются. Понятие алгоритма в информатике является фундаментальным.

Таким же, какими являются понятия точки, прямой и плоскости в геометрии, вещества в химии, пространства и времени в физике и т. д.

Алгоритм. Свойства алгоритма. Способы описания алгоритма

Свойства алгоритма:

- дискретность (прерывность, раздельность) - алгоритм должен представлять процесс решения задачи как последовательное выполнение простых шагов (этапов);
- определенность - каждый шаг алгоритма должен быть четким и однозначным. Выполнение алгоритма носит механический характер и не требует никаких дополнительных сведений о решаемой задаче;
- результативность - алгоритм должен приводить к решению задачи за конечное число шагов;
- массовость - алгоритм решения разрабатывается в общем виде, т. е. он должен быть применим для решения некоторого класса задач, различающихся лишь исходными данными.

Алгоритм. Свойства алгоритма. Способы описания алгоритма

Способы описания алгоритмов

- словесный;
- графический;
- табличный;
- формульный.

Алгоритм. Свойства алгоритма. Способы описания алгоритма

Словесный способ каждый из нас использует ежедневно, пересказывая собеседнику, например, различные инструкции, правила, кулинарные рецепты, т. е. какую-то последовательность, приводящую к конечному результату.

Алгоритм. Свойства алгоритма. Способы описания алгоритма

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным. Как часто для лучшего понимания той или иной ситуации нам проще начертить какую-то схему, план, согласно которым мы будем действовать.

В программировании данный способ предпочтителен, поскольку позволяет с помощью последовательности функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий, представить ход решения той или иной задачи.

Такое представление алгоритма называется структурной схемой алгоритма, или блок-схемой.

Алгоритм. Свойства алгоритма. Способы описания алгоритма

Табличный способ используется, например, в бухгалтерии при составлении ежегодных отчетов, сводок и т. д.

Алгоритм. Свойства алгоритма. Способы описания алгоритма

Формульный способ находит свое применение при решении задач из области математики, физики и т. д. Например, при решении квадратного уравнения мы приступаем к нахождению дискриминанта уравнения, а затем, в зависимости от полученного результата, находим корни уравнения по известным всем формулам.

Блок-схемы алгоритмов

Блок-схема — распространённый тип схем (графических моделей), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединённых между собой линиями, указывающими направление последовательности.

Элементы блок-схем алгоритмов

Ограничитель

Символ отображает вход из внешней среды и выход во внешнюю среду (начало или конец схемы программы, внешнее использование и источник или пункт назначения данных).

На практике имеют смысл следующие описания ограничителей: начало/конец, запуск/останов, перезапуск (подразумевает перезапуск данной блок-схемы), ошибка (подразумевает завершение алгоритма с ошибкой), исключение (подразумевает исполнение программного исключения)

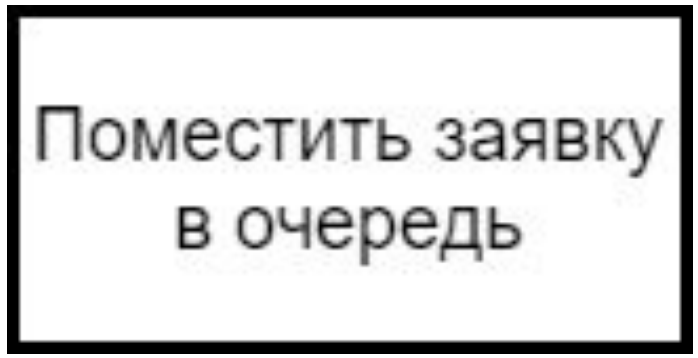
Элементы блок-схем алгоритмов

Данные (ввод-вывод)

Символ отображает данные, носитель данных не определён.

Преобразование данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). Данный символ не определяет носителя данных (для указания типа носителя данных используются специфические символы).

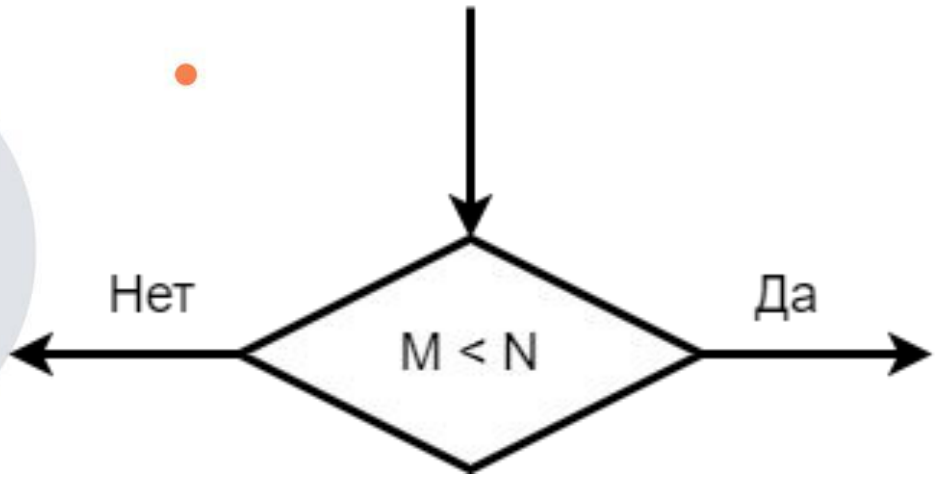
Элементы блок-схем алгоритмов



Действие

Символ отображает функцию обработки данных любого вида (выполнение определённой операции или группы операций, приводящее к изменению значения, формы или размещения информации или к определению, по которому из нескольких направлений потока следует двигаться).

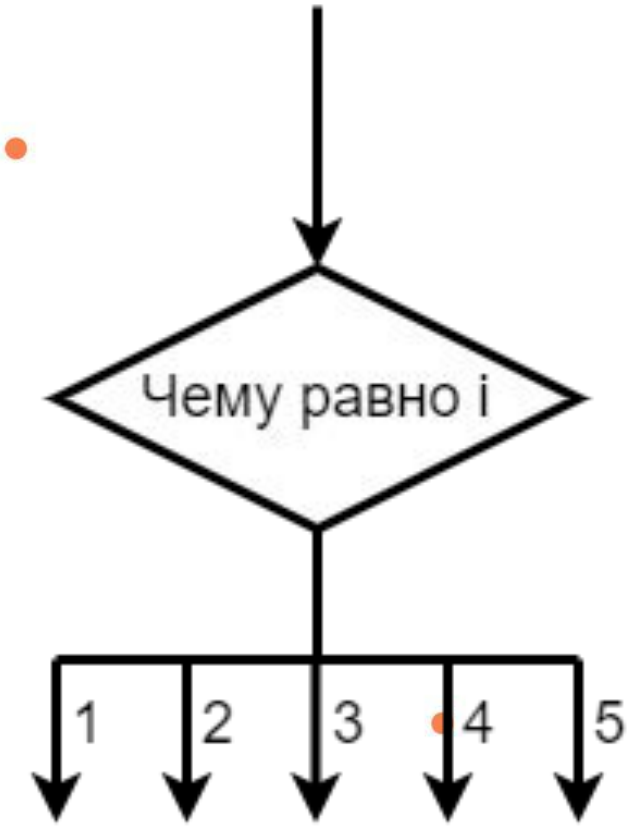
Элементы блок-схем алгоритмов



Вопрос (условие или решение)

Отображает решение или функцию переключательного типа с одним входом и двумя или более альтернативными выходами, из которых только один может быть выбран после вычисления условий, определённых внутри этого элемента. Вход в элемент обозначается линией, входящей обычно в верхнюю вершину элемента.

Элементы блок-схем алгоритмов



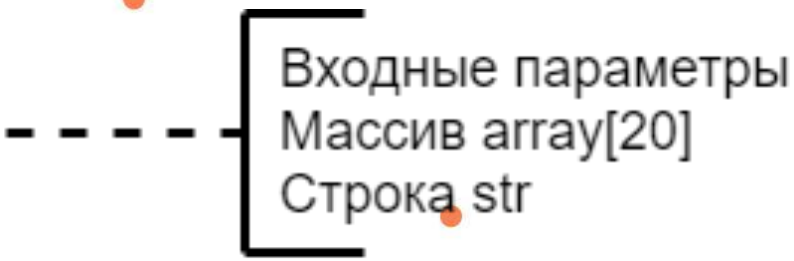
Если выходов два или три, то обычно каждый выход обозначается линией, выходящей из оставшихся вершин (боковых и нижней). Если выходов больше трёх, то их следует показывать одной линией, выходящей из вершины (чаще нижней) элемента, которая затем разветвляется. Соответствующие результаты вычислений могут записываться рядом с линиями, отображающими эти пути. Примеры решения: в общем случае — сравнение (три выхода: >, <, =); в программировании — условные операторы if (два выхода: true, false) и case (множество выходов).

Элементы блок-схем алгоритмов

Предопределённый процесс (функция)

Символ отображает предопределённый процесс, состоящий из одной или нескольких операций или шагов программы, которые определены в другом месте (в подпрограмме, модуле). Например, в программировании – вызов процедуры или функции.

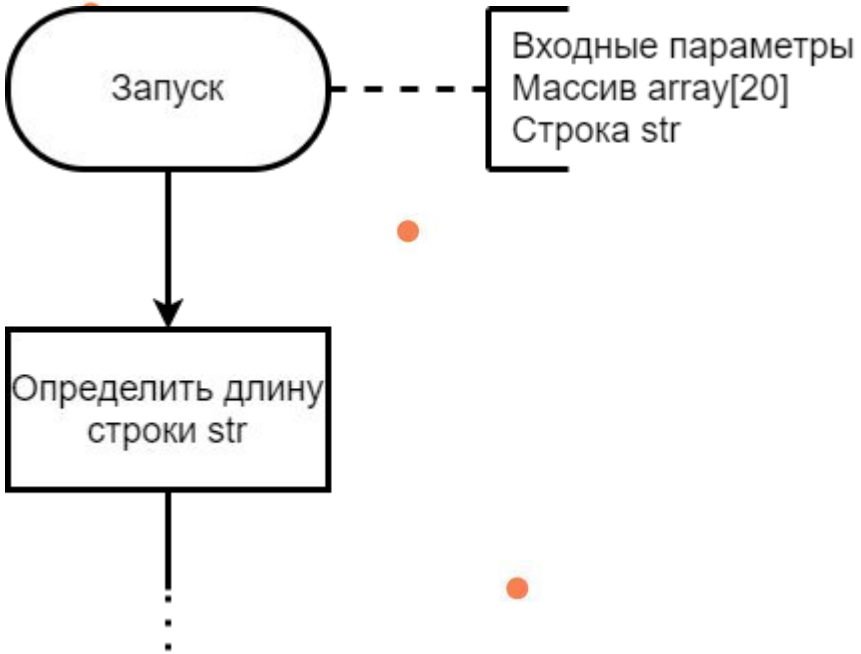
Элементы блок-схем алгоритмов



Комментарий

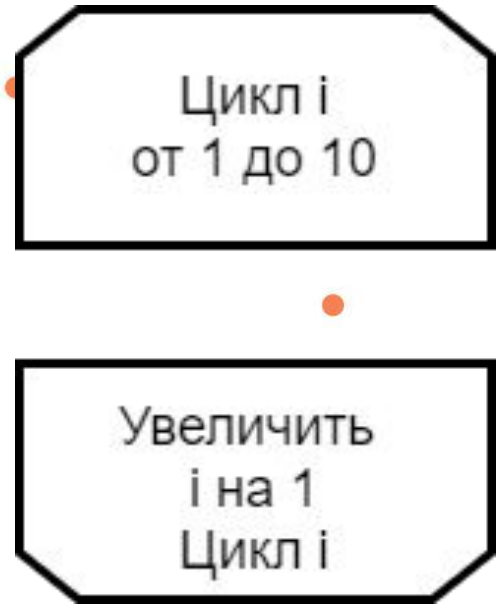
Символ используют для добавления описательных комментариев или пояснительных записей в целях объяснения или примечаний. Пунктирные линии в символе комментария связаны с соответствующим символом или могут обводить группу символов. Текст комментариев или примечаний должен быть помещён около ограничивающей фигуры.

Элементы блок-схем алгоритмов



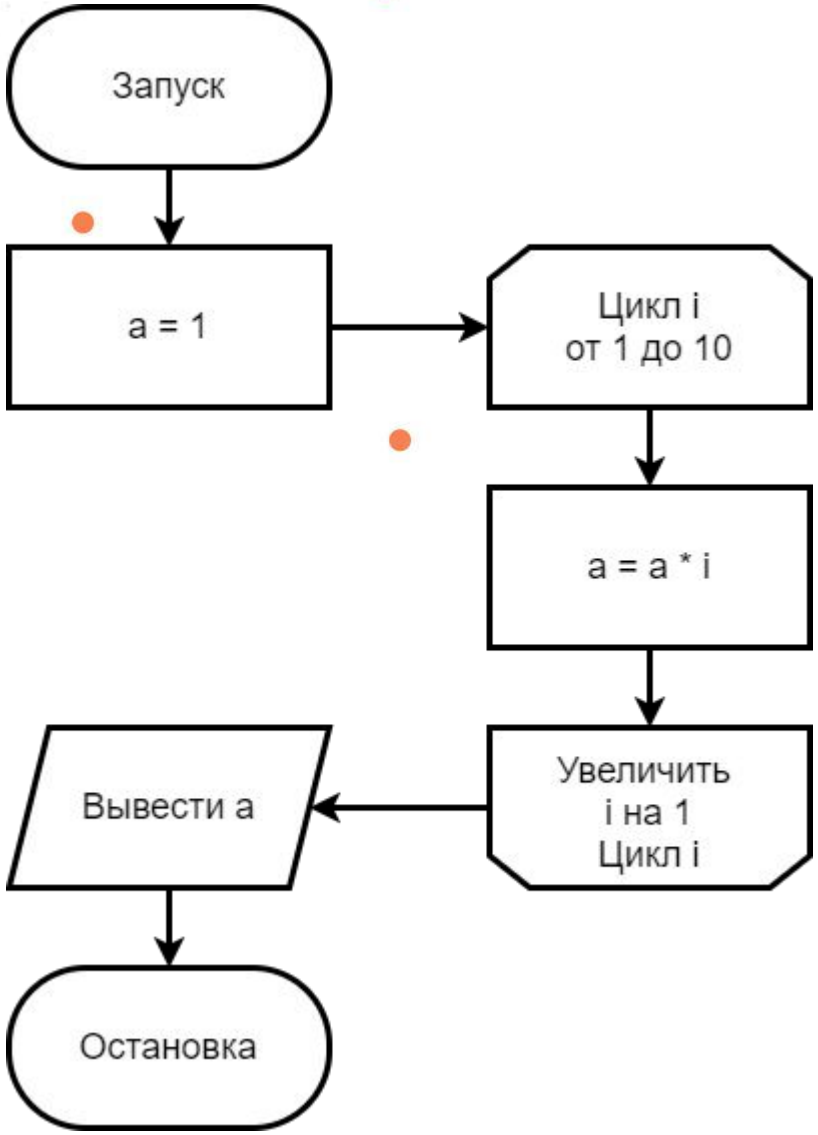
Также символ комментария следует использовать в тех случаях, когда объём текста, помещаемого внутри некоего символа (например, символ процесса, символ данных и др.), превышает размер самого этого символа. Комментарии используют совместно с терминаторами для описания входных аргументов алгоритма при описании функций

Элементы блок-схем алгоритмов



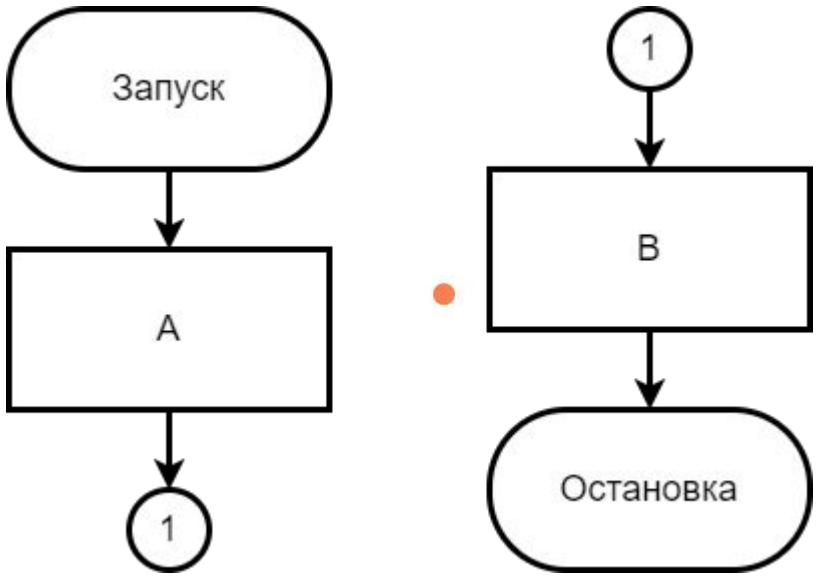
Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т. д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие.

Элементы блок-схем алгоритмов



Пример применения цикла.

Элементы блок-схем алгоритмов



Соединитель

Символ отображает выход в часть схемы и вход из другой части этой схемы и используется для обрыва линии и продолжения её в другом месте. Соответствующие символы-соединители должны содержать одно и то же уникальное обозначение.

Введение в Python

Python представляет популярный высокоуровневый язык программирования, который предназначен для создания приложений различных типов. Это и веб-приложения, и игры, и настольные программы, и работа с базами данных. Довольно большое распространение питон получил в области машинного обучения и исследований искусственного интеллекта.

Впервые язык Python был анонсирован в 1991 году голландским разработчиком Гвидо Ван Россумом. С тех пор данный язык проделал большой путь развития. В 2000 году была издана версия 2.0, а в 2008 году - версия 3.0.

Введение в Python

Основные особенности языка программирования Python:

- Скриптовый язык. Код программ определяется в виде скриптов.
- Поддержка самых различных парадигм программирования, в том числе объектно-ориентированной и функциональной парадигм.
- Интерпретация программ. Для работы со скриптами необходим интерпретатор, который запускает и выполняет скрипт.
- Портативность и платформонезависимость. Не имеет значения, какая у нас операционная система - Windows, Mac OS, Linux, нам достаточно написать скрипт, который будет запускаться на всех этих ОС при наличии интерпретатора
- Автоматическое управление памяти
- Динамическая типизация

Введение в Python

**Пояснение:*

Сценарный язык (язык сценариев, скриптовой язык; англ. scripting language) — высокоуровневый язык сценариев (англ. script) — кратких описаний действий, выполняемых системой. Разница между программами и сценариями довольно размыта. Сценарий — это программа, имеющая дело с готовыми программными компонентами, которые, однажды загруженные, в своей работе не зависят от дальнейшего наличия/отсутствия подключения к Сети.

Согласно Джону Устерхауту, автору языка Tcl, высокоуровневые языки можно разделить на языки системного программирования (англ. system programming languages) и сценарные языки (англ. scripting languages). Последние он также назвал склеивающими языками (англ. glue languages) или языками системной интеграции (англ. system integration languages). Сценарии обычно интерпретируются, а не компилируются.

Введение в Python

Выполнение программы на Python выглядит следующим образом. Сначала мы пишем в текстовом редакторе скрипт с набором выражений на данном языке программирования. Передаем этот скрипт на выполнение интерпретатору. Интерпретатор транслирует код в промежуточный байткод, а затем виртуальная машина переводит полученный байткод в набор инструкций, которые выполняются операционной системой.

Введение в Python

Здесь стоит отметить, что хотя формально трансляция интерпретатором исходного кода в байткод и перевод байткода виртуальной машиной в набор машинных команд представляют два разных процесса, но фактически они объединены в самом интерпретаторе.

Введение в Python

Python - очень простой язык программирования, он имеет лаконичный и в то же время довольно простой и понятный синтаксис. Соответственно его легко изучать, и собственно это одна из причин, по которой он является одним из самых популярных языков программирования именно для обучения. В частности, в 2014 году он был признан самым популярным языком программирования для обучения в США.

Python также популярен не только в сфере обучения, но в написании конкретных программ в том числе коммерческого характера. В немалой степени поэтому для этого языка написано множество библиотек, которые мы можем использовать.

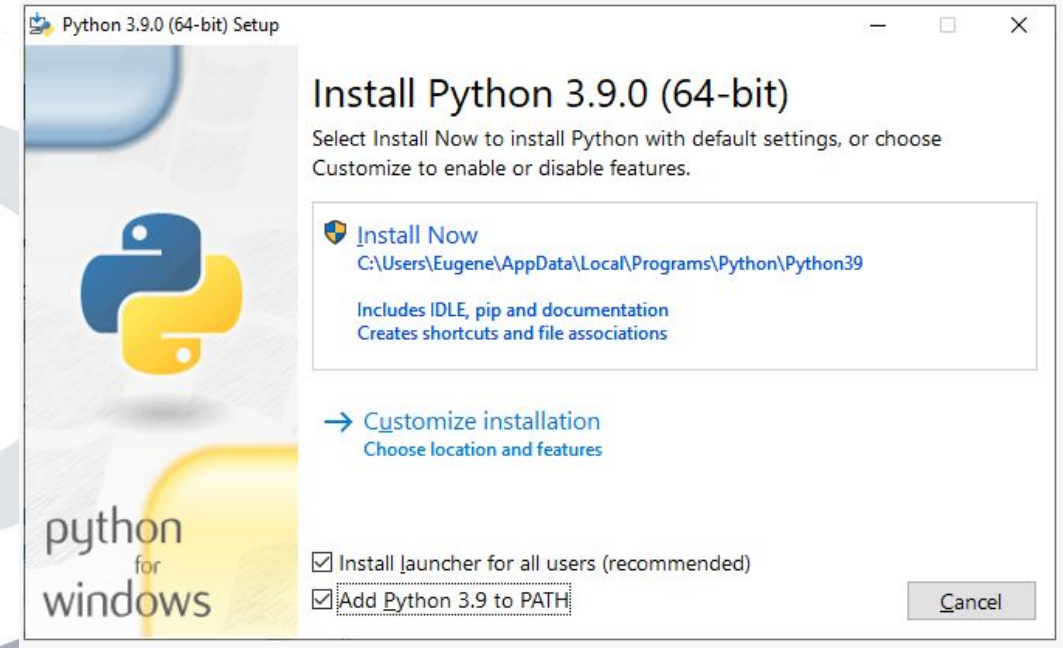
Кроме того, у данного языка программирования очень большое коммьюнити, в интернете можно найти по данному языку множество полезных материалов, примеров, получить квалифицированную помощь специалистов.

Введение в Python

Установка Python

Для создания программ на Python нам потребуется интерпретатор. Для его установки перейдем на страницу https://www.python.org/downloads/

Введение в Python



По нажатию на кнопку будет загружен соответствующей текущей ОС установщик Python.

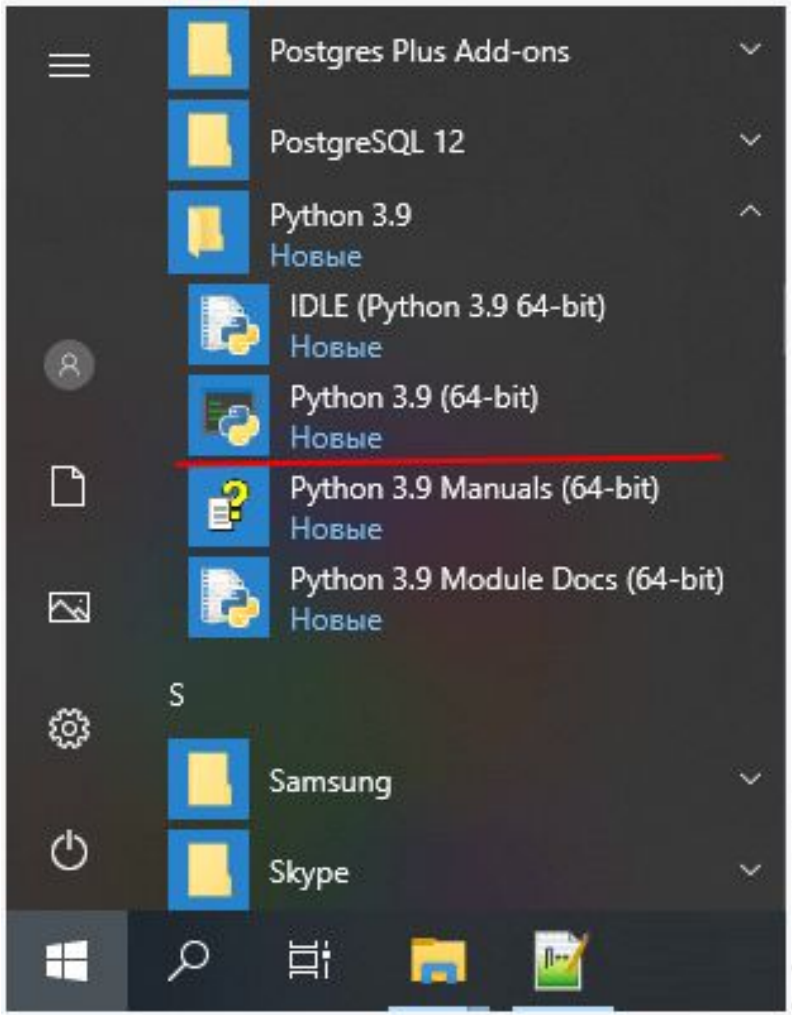
На ОС Windows при запуске инсталлятора запускает окно мастера установки:

Введение в Python

Здесь мы можем задать путь, по которому будет устанавливаться интерпретатор. Оставим его по умолчанию, то есть C:\Users\[имя_пользователя]\AppData\Local\Programs\Python\Python36\.

Кроме того, в самом низу отметим флажок "Add Python 3.9 to PATH", чтобы добавить путь к интерпретатору в переменные среды.

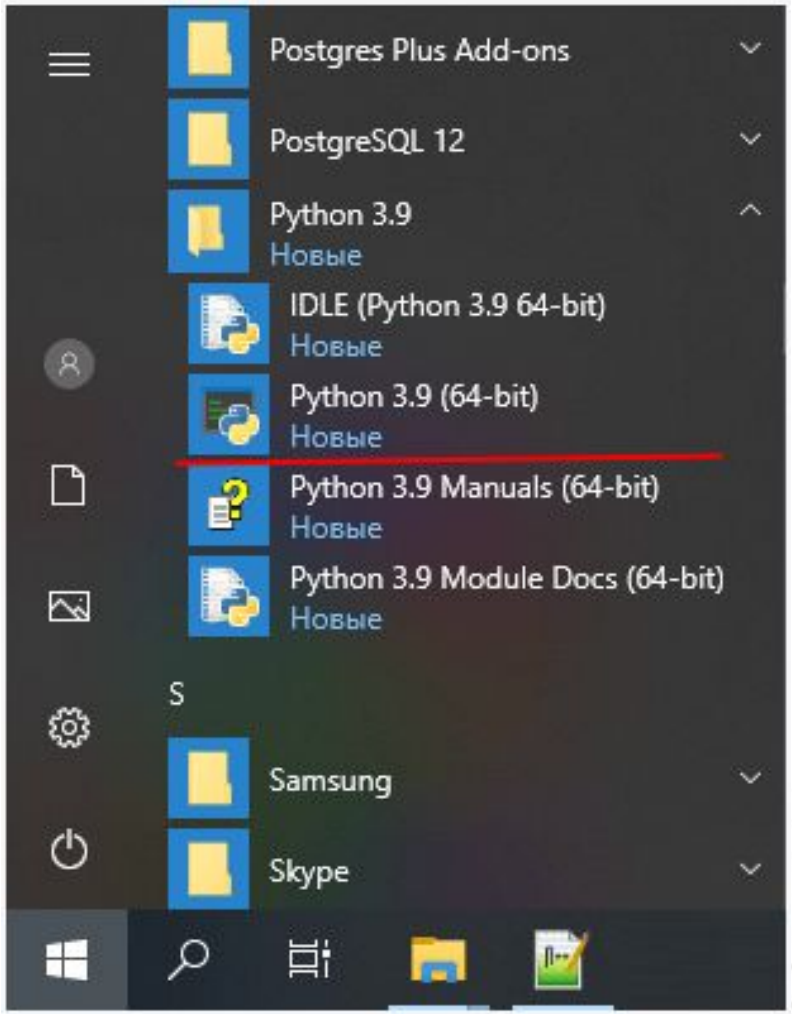
После установки в меню Пуск на ОС Windows мы сможем найти иконки для доступа к разным утилитам питона:



Введение в Python

Здесь утилита Python 3.9 (64-bit) представляет интерпретатор, в котором мы можем запустить скрипт. В файловой системе сам файл интерпретатора можно найти по пути, по которому производилась установка.

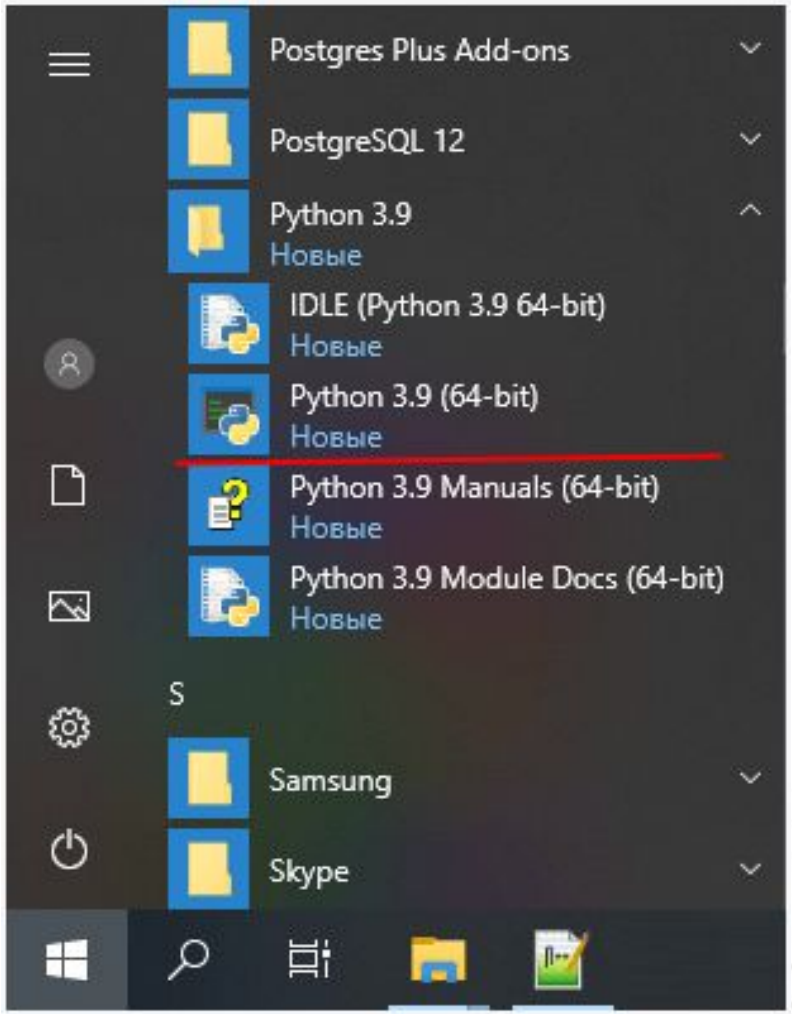
На Windows по умолчанию это путь C:\Users\[имя_пользователя]\AppData\Local\Programs\Python\Python37, а сам интерпретатор представляет файл python.exe. На ОС Linux установка производится по пути /usr/local/bin/python3.9.



Первая программа на Python

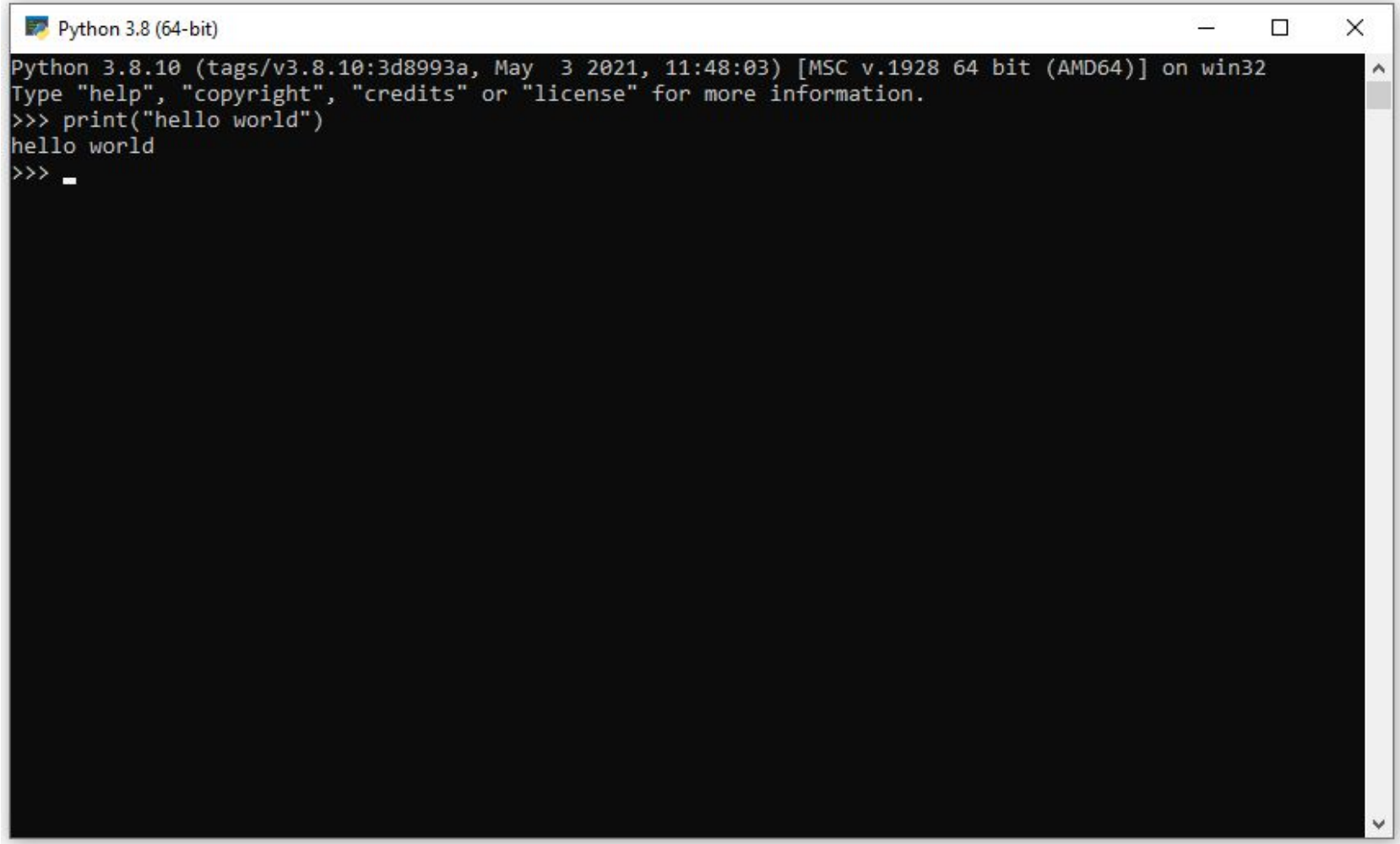
После установки интерпретатора, как было описано в выше, мы можем начать создавать приложения на Python.

По умолчанию (на Windows) он будент установлен по пути C:\Users\[имя_пользователя]\AppData\Local\Programs\Python\Python39\ и представляет файл под названием python.exe.



Первая программа на Python

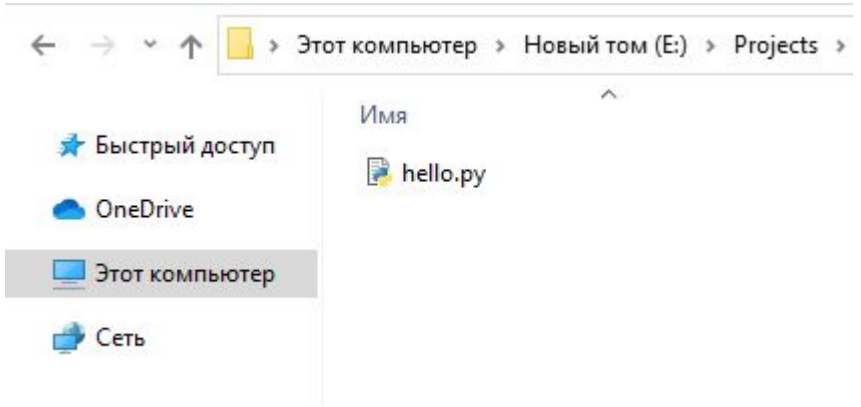
Для этой программы использовалась функция **print()**, которая выводит некоторую строку на консоль.



Первая программа на Python

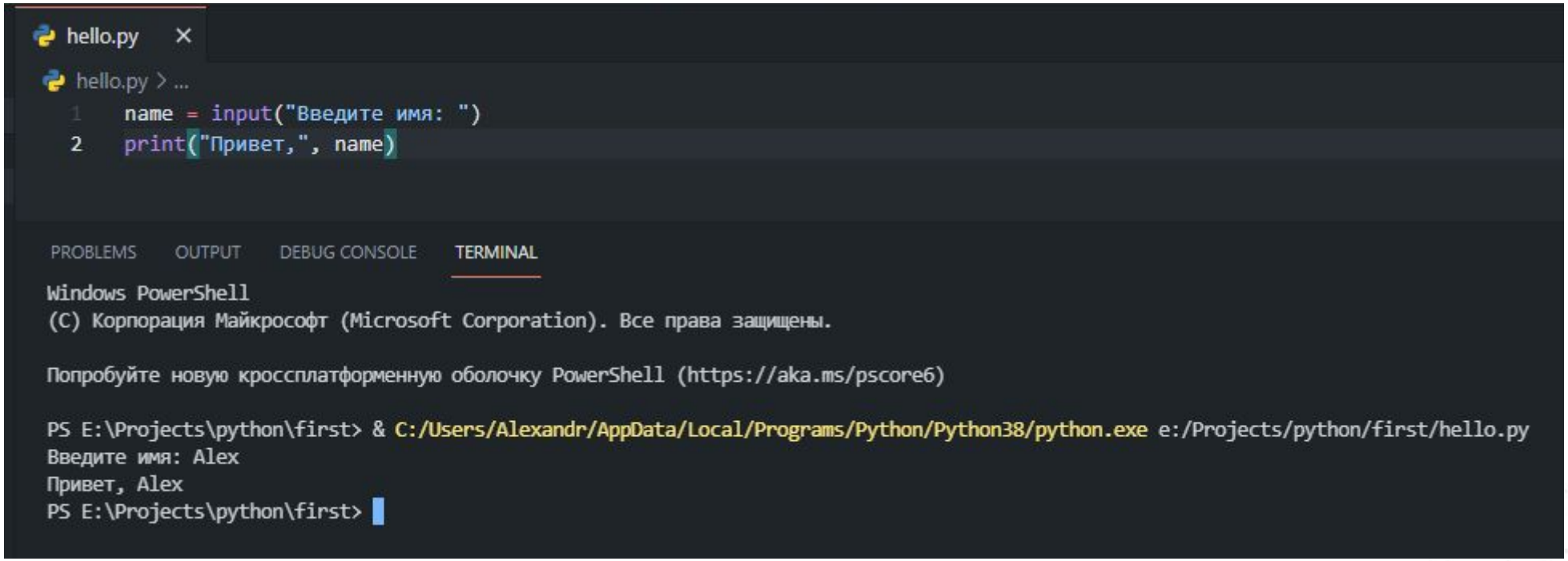
Создание файла программы

В реальности, как правило, программы определяются во внешних файлах-скриптах и затем передаются интерпретатору на выполнение. Поэтому создадим файл программы. Для этого на диске C или где-нибудь в другом месте файловой системы определим для скриптов папку python. А в этой папке создадим новый текстовый файл, который назовем hello.py. По умолчанию файлы с кодом на языке Python, как правило, имеют расширение py.



Первая программа на Python

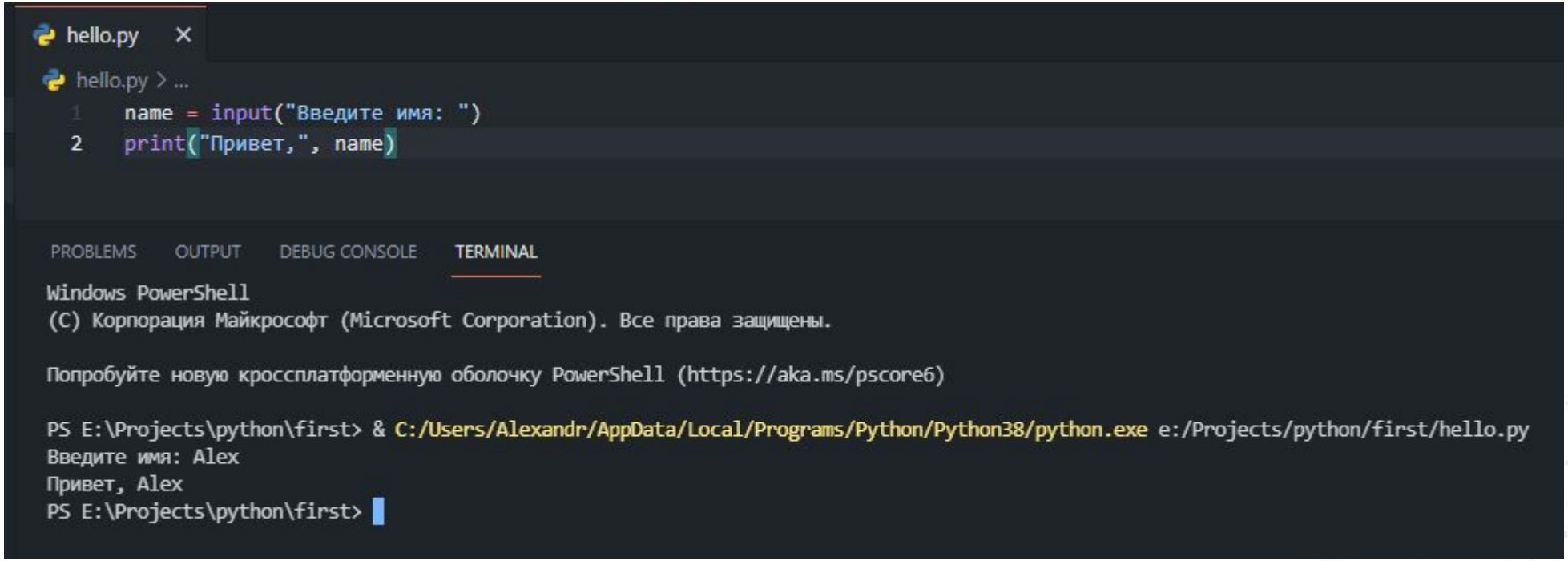
Откроем этот файл в любом текстовом редакторе и добавим в него следующий код:



Первая программа на Python

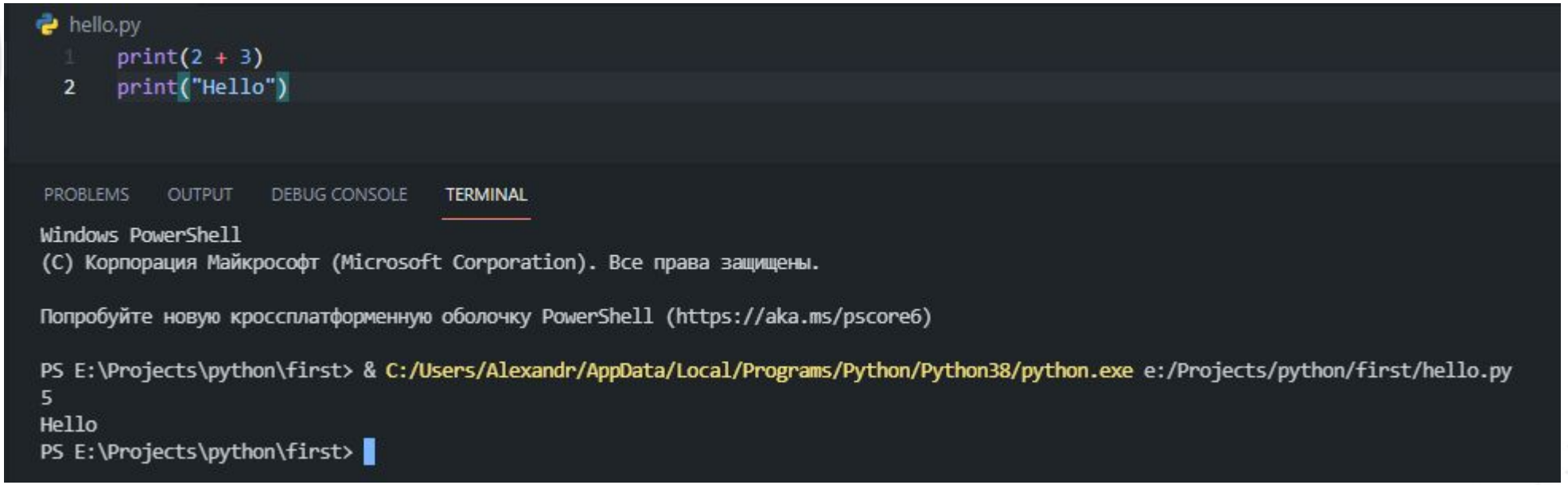
Скрипт состоит из двух строк. Первая строка с помощью функции input() ожидает ввода пользователем своего имени. Введенное имя затем попадает в переменную name.

Вторая строка с помощью функции print() выводит приветствие вместе с введенным именем.



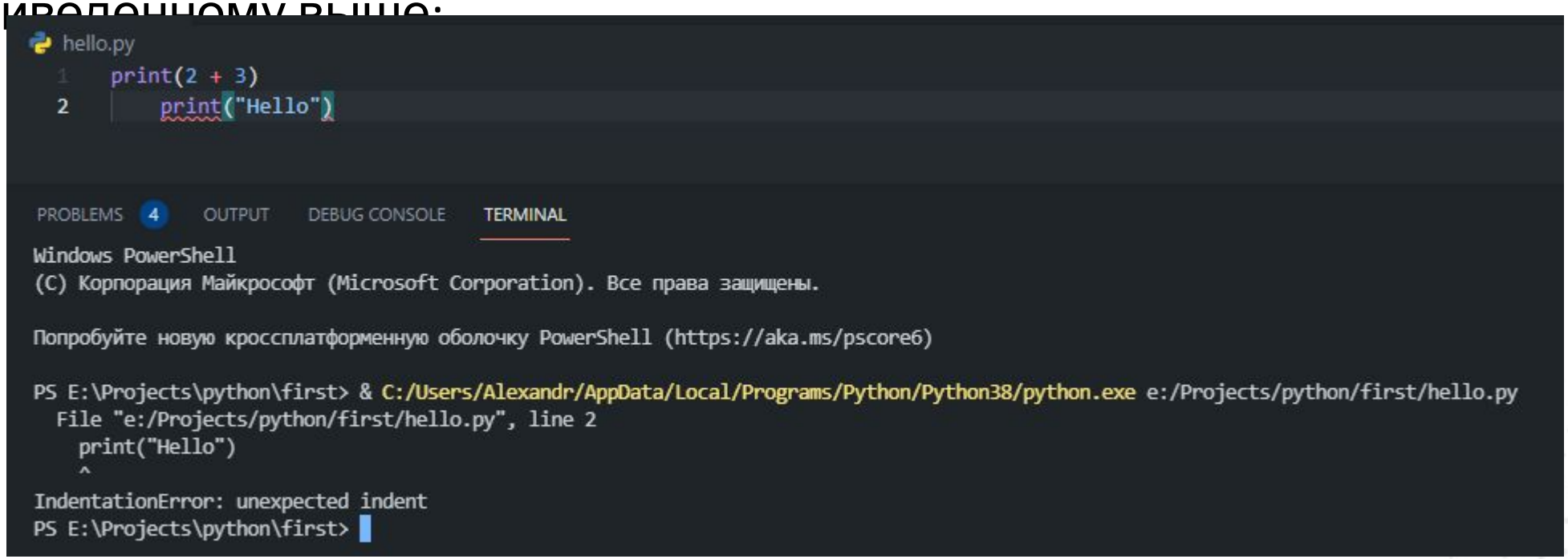
Введение в написание программ

Программа на языке Python состоит из набора инструкций. Каждая инструкция помещается на новую строку. Например:



Введение в написание программ

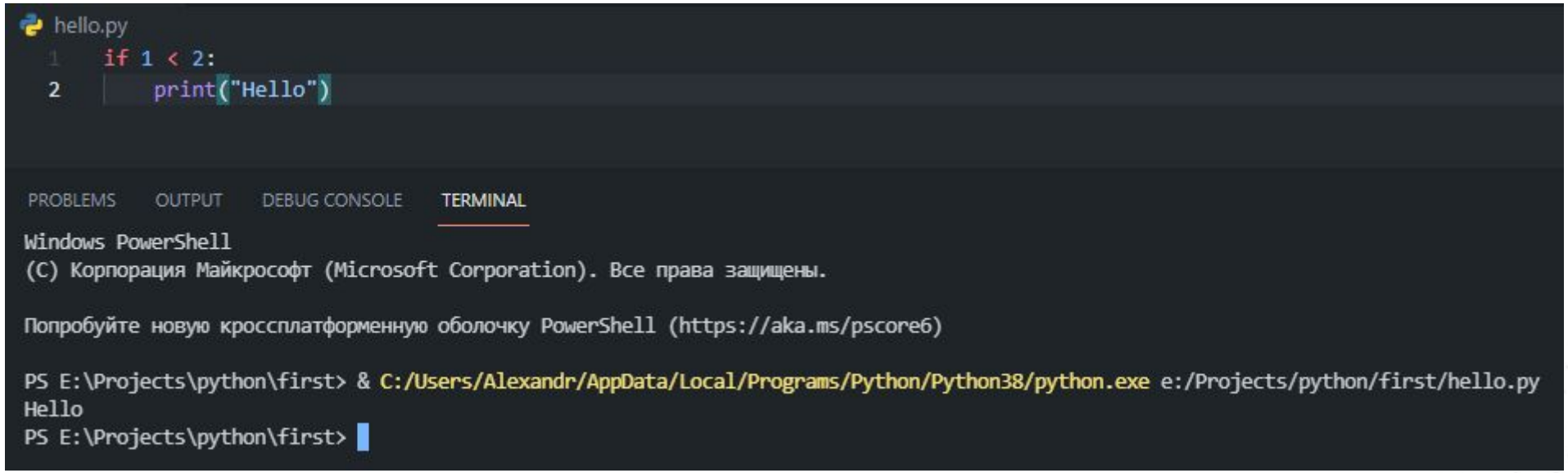
Большую роль в Python играют отступы. Неправильно поставленный отступ фактически является ошибкой. Например, в следующем случае мы получим ошибку, хотя код будет практически аналогичен приведенному выше:



Введение в написание программ

Поэтому стоит помещать новые инструкции сначала строки. В этом одно из важных отличий пайтона от других языков программирования, как C# или Java.

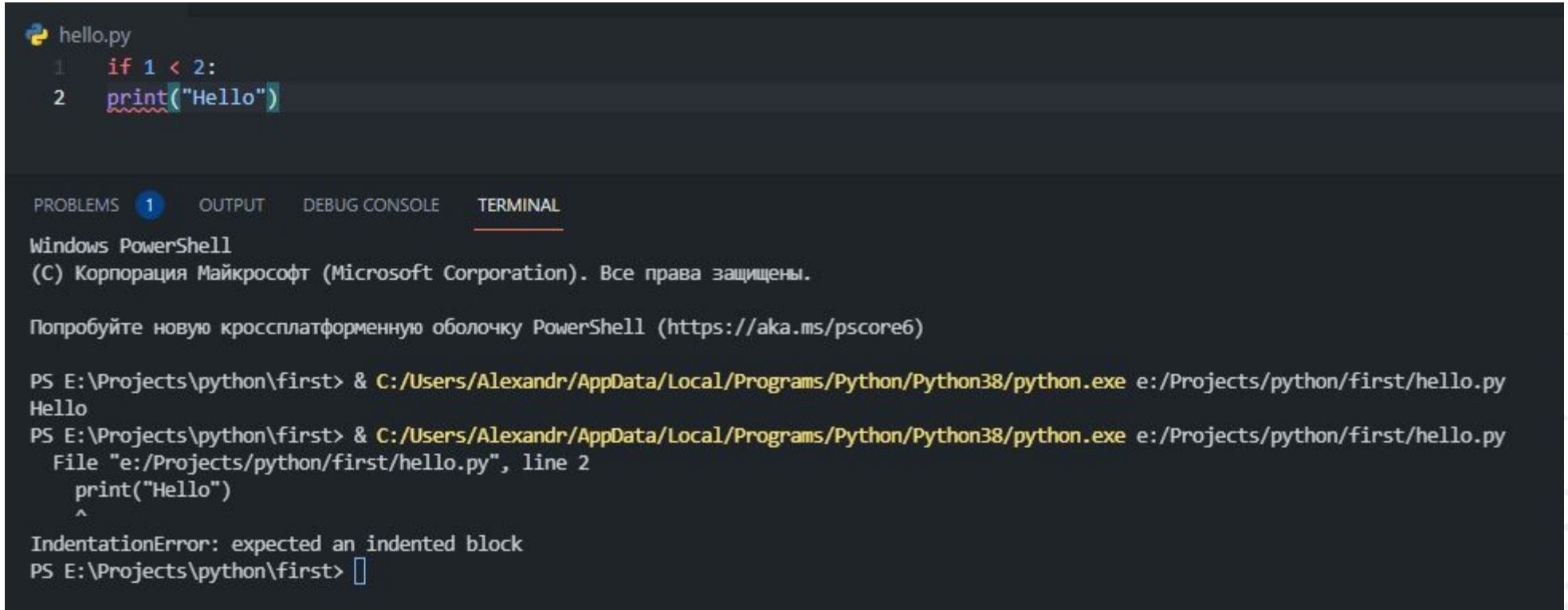
Однако стоит учитывать, что некоторые конструкции языка могут состоять из нескольких строк. Например, условная конструкция if:



Введение в написание программ

В данном случае если 1 меньше 2, то выводится строка "Hello". И здесь уже должен быть отступ, так как инструкция print("Hello") используется не сама по себе, а как часть условной конструкции if. Причем отступ, согласно руководству по оформлению кода, желательно делать из такого количество пробелов, которое кратно 4 (то есть 4, 8, 16 и т.д.) Хотя если отступов будет не 4, а 5, то программа также будет работать.

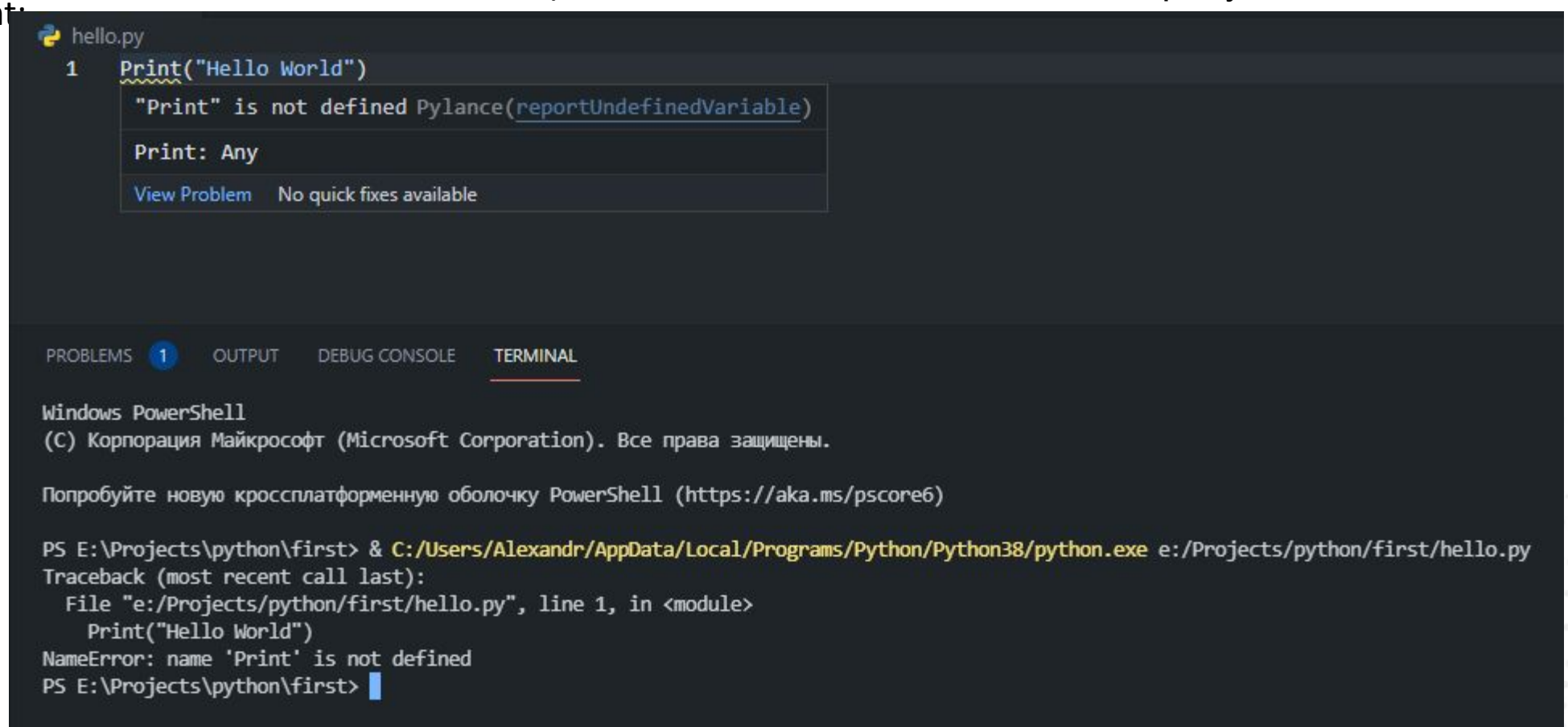
Таких конструкций не так много, поэтому особой путаницы по поводу где надо, а где не надо ставить пробелы, не должно возникнуть.



Введение в написание программ

Регистрозависимость

Python - регистрозависимый язык, поэтому выражения print и Print или PRINT представляют разные выражения. И если вместо метода print для вывода на консоль мы попробуем использовать метод Print:



Введение в написание программ

Комментарии

Для отметки, что делает тот или иной участок кода, применяются комментарии. При трансляции и выполнении программы интерпретатор игнорирует комментарии, поэтому они не оказывают никакого влияния на работу программы.

Комментарии в Python бывают блочные и строчные. Все они предваряются знаком решетки (#).

hello.py

```
1   # Блочные комментарии ставятся в начале строки
2   print("Hello World")
3
4   print("Hello World")  # Строчные комментарии располагаются на той же строке, что и инструкции языка
```

Введение в написание программ

Основные функции

Python предоставляет ряд встроенных функций. Некоторые из них используются очень часто, особенно на начальных этапах изучения языка, поэтому рассмотрим их. Основной функцией для вывода информации на консоль является функция **print()**.

hello.py

```
# В качестве аргумента в эту функцию
# передается строка, которую мы хотим вывести
print("Hello Python")

# Если же нам необходимо вывести несколько значений на консоль,
# то мы можем передать их в функцию print через запятую
print("Full name:", "Tom", "Smith")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Hello Python
Full name: Tom Smith
PS E:\Projects\python\first>
```

Введение в написание программ

Если функция print отвечает за вывод, то функция input отвечает за ввод информации. В качестве необязательного параметра эта функция принимает приглашение к вводу и возвращает введенную строку, которую мы можем сохранить в переменную:

hello.py > ...

```
1   a = input("Введите a: ")
2   b = input("Введите b: ")
3   print(a + b)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

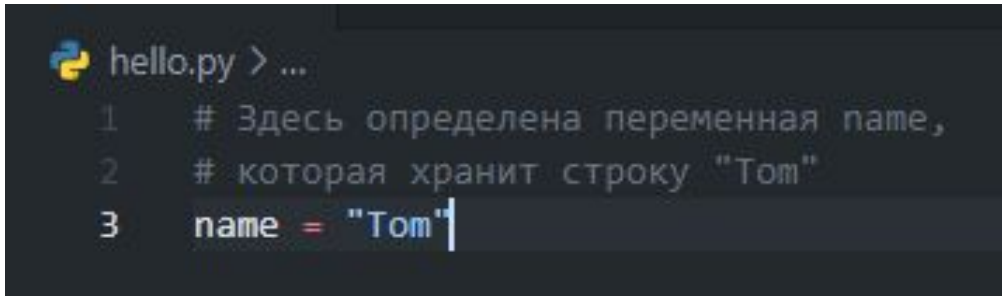
```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Hello Python
Full name: Tom Smith
PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Введите a: 1
Введите b: 2
12
PS E:\Projects\python\first>
```

Переменные и типы данных

Переменная хранит определенные данные. Название переменной в Python должно начинаться с алфавитного символа или со знака подчеркивания и может содержать алфавитно-цифровые символы и знак подчеркивания. И кроме того, название переменной не должно совпадать с названием ключевых слов языка Python. Ключевых слов не так много, их легко запомнить: **and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield.**

Переменные и типы данных

Например, создадим переменную:



Переменные и типы данных

В пайтоне применяется два типа наименования переменных: camel case и underscore notation (snake case). И также надо учитывать регистрозависимость, поэтому переменные name и Name будут представлять разные объекты.

hello.py > ...

```
1   # Camel case подразумевает, что каждое новое подслово
2   # в наименовании переменной начинается с большой буквы
3   userName = "Tom"
4
5   # Underscore notation подразумевает, что подслова
6   # в наименовании переменной разделяются знаком подчеркивания
7   user_name = "Tom"
```

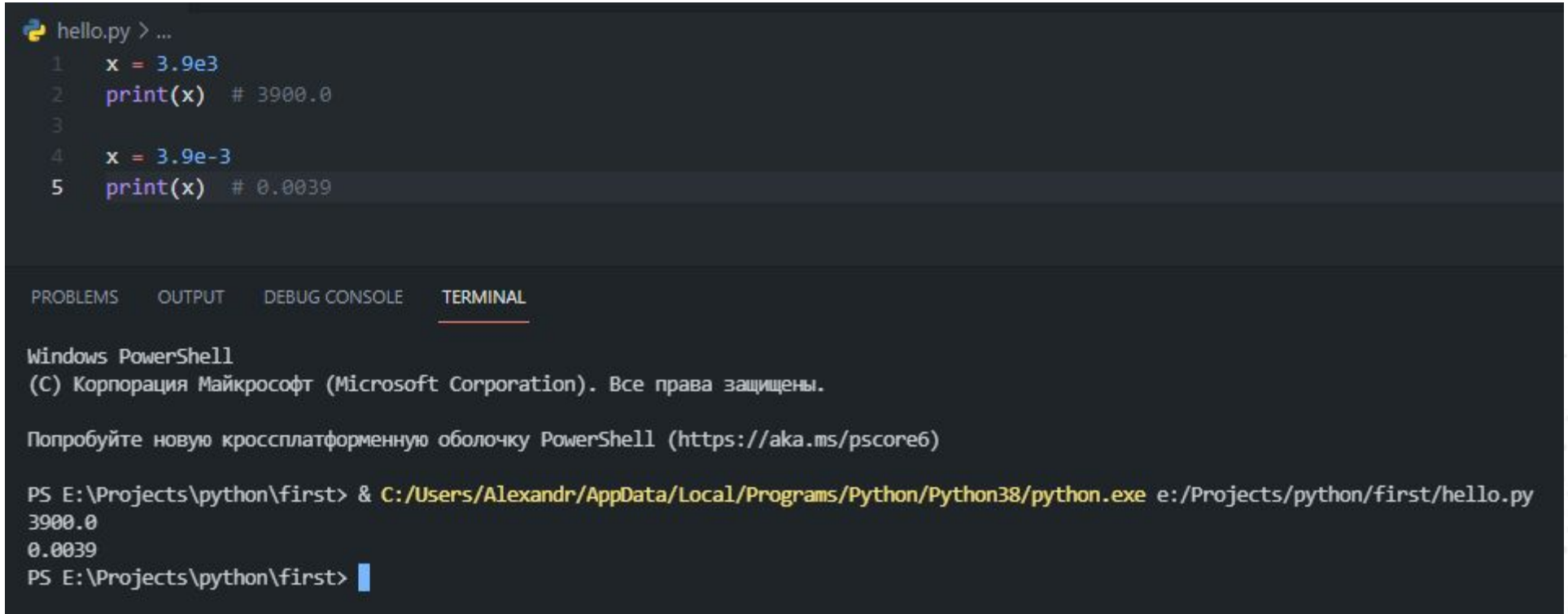
Переменные и типы данных

Переменная хранит данные одного из типов данных. В Python существует множество различных типов данных, которые подразделяются на категории: числа, последовательности, словари, наборы:

- boolean - логическое значение True или False
- int - представляет целое число, например, 1, 4, 8, 50.
- float - представляет число с плавающей точкой, например, 1.2 или 34.76
- complex - комплексные числа
- str - строки, например "hello". В Python 3.x строки представляют набор символов в кодировке Unicode
- bytes - последовательность чисел в диапазоне 0-255
- byte array - массив байтов, аналогичен bytes с тем отличием, что может изменяться
- list - список
- tuple - кортеж
- set - неупорядоченная коллекция уникальных объектов
- frozen set - то же самое, что и set, только не может изменяться (immutable)
- dict - словарь, где каждый элемент имеет ключ и значение

Переменные и типы данных

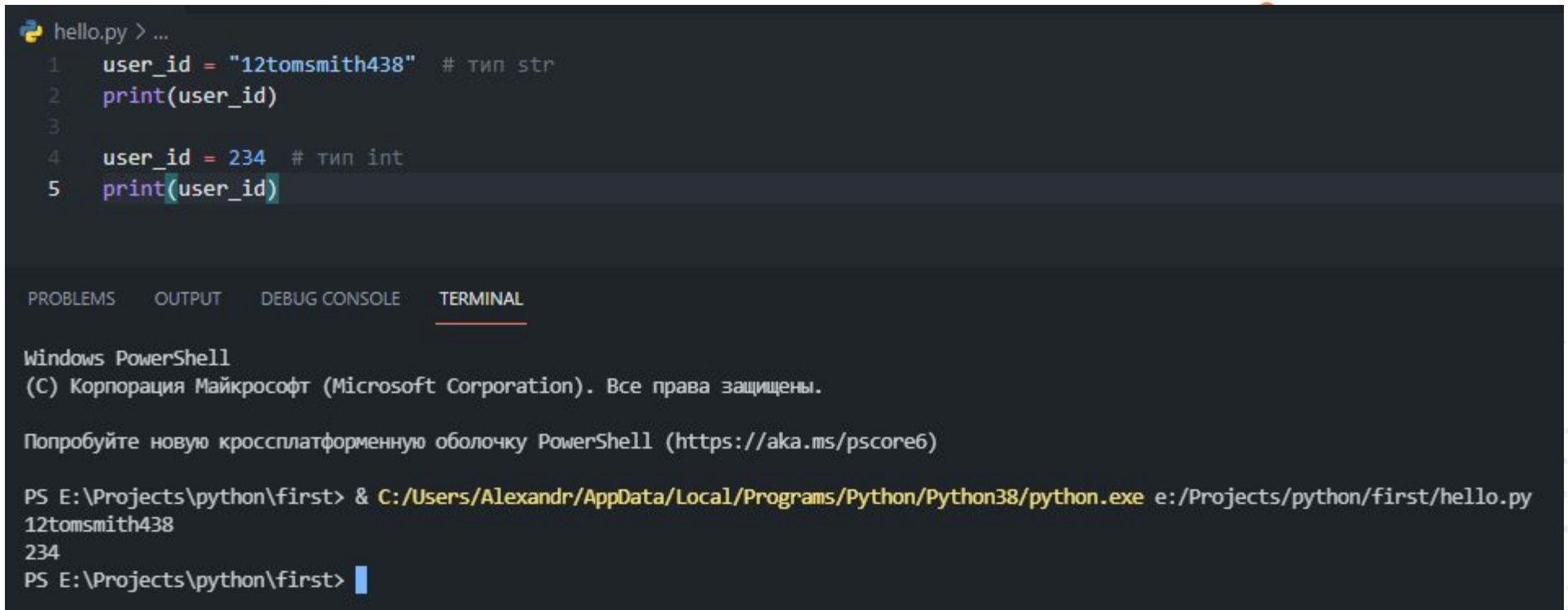
Python является языком с динамической типизацией. Он определяет тип данных переменной исходя из значения, которое ей присвоено. Так, при присвоении строки в двойных или одинарных кавычках переменная имеет тип str. При присвоении целого числа Python автоматически определяет тип переменной как int. Чтобы определить переменную как объект float, ей присваивается дробное число, в котором разделителем целой и дробной части является точка. Число с плавающей точкой можно определять в экспоненциальной записи:



Переменные и типы данных

Число float может иметь только 18 значимых симолов. Так, в данном случае используются только два символа - 3.9. И если число слишком велико или слишком мало, то мы можем записывать число в подобной нотации, используя экспоненту. Число после экспоненты указывает степень числа 10, на которое надо умножить основное число - 3.9.

При этом в процессе работы программы мы можем изменить тип переменной, присвоив ей значение другого типа:



Переменные и типы данных

С помощью функции type() динамически можно узнать текущий тип переменной:

hello.py > ...

```python
user_id = "12tomsmith438"
print(type(user_id))  # <class 'str'>

user_id = 234
print(type(user_id))  # <class 'int'>
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
<class 'str'>
<class 'int'>
PS E:\Projects\python\first>
```

Операции с числами

Арифметические операции

Python поддерживает все распространенные арифметические операции:

hello.py

```python
print(6 + 2)  # Сложение двух чисел: 8

print(6 - 2)  # Вычитание двух чисел: 4

print(6 * 2)  # Умножение двух чисел: 12

print(7 / 2)  # Деление двух чисел: 3.5

print(7 // 2)  # Целочисленное деление двух чисел: 3

print(6 ** 2)  # Возведение в степень: возводим число 6 в степень 2. Результат - 36

# Получение остатка от деления числа 7 на 2. Результат - 1
# В данном случае ближайшее число к 7, которое делится на 2 без остатка, это 6
# Поэтому остаток от деления равен 7 - 6 = 1
print(7 % 2)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Pr
8
4
12
3.5
3
36
1
PS E:\Projects\python\first>
```

Операции с числами

При последовательном использовании нескольких арифметических операций их выполнение производится в соответствии с их приоритетом. В начале выполняются операции с большим приоритетом. Приоритеты операций в порядке убывания приведены в следующей таблице.

Операции	Направление
**	Справа налево
* / // %	Слева направо
+ -	Слева направо

Операции с числами

Пусть у нас выполняется следующее выражение:

hello.py > ...

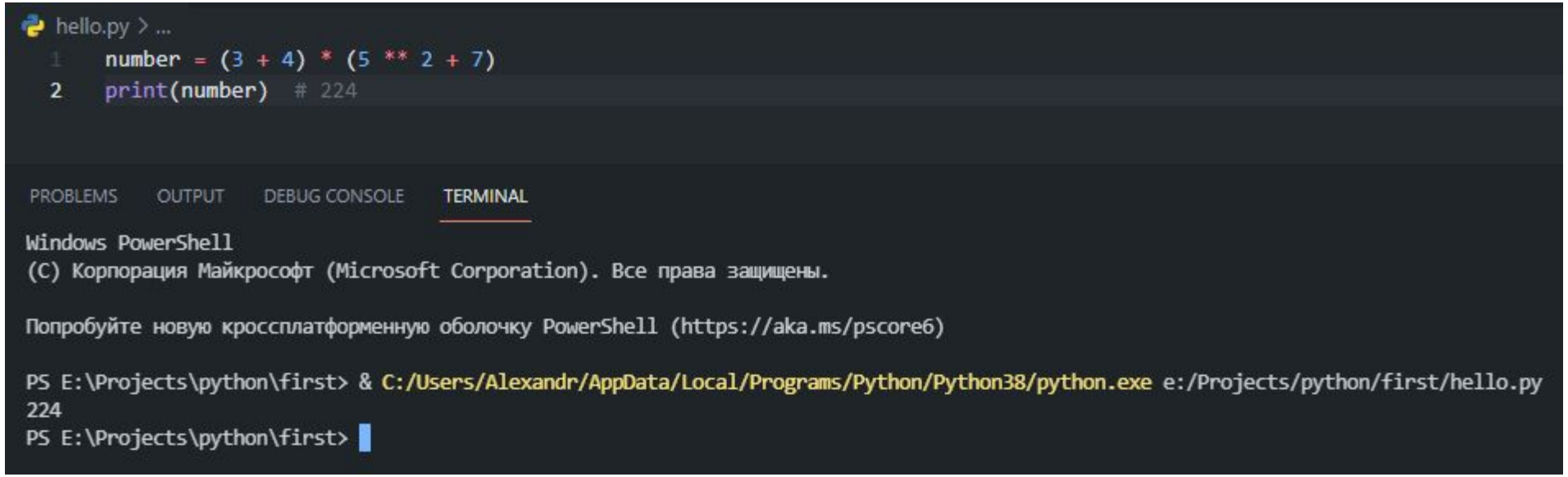
```python
# Здесь начале выполняется возведение в степень (5 ** 2) как операция с большим приоритетом,
# далее результат умножается на 4 (25 * 4), затем происходит сложение (3 + 100)
# и далее опять идет сложение (103 + 7)
number = 3 + 4 * 5 ** 2 + 7
print(number)  # 110
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
110
PS E:\Projects\python\first>
```

Операции с числами

Чтобы переопределить порядок операций, можно использовать скобки:



Следует отметить, что в арифметических операциях могут принимать участие как целые, так и дробные числа. Если в одной операции участвует целое число (int) и число с плавающей точкой (float), то целое число приводится к типу float.

Операции с числами

Арифметические операции с присвоением

Ряд специальных операций позволяют использовать присвоить результат операции первому операнду:

+= Присвоение результата сложения

-= Присвоение результата вычитания

*= Присвоение результата умножения

/= Присвоение результата от деления

//= Присвоение результата целочисленного деления

**= Присвоение степени числа

%= Присвоение остатка от деления

Операции с числами

Примеры операций:

hello.py > ...

```python
number = 10
number += 5
print(number)  # 15

number -= 3
print(number)  # 12

number *= 4
print(number)  # 48
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

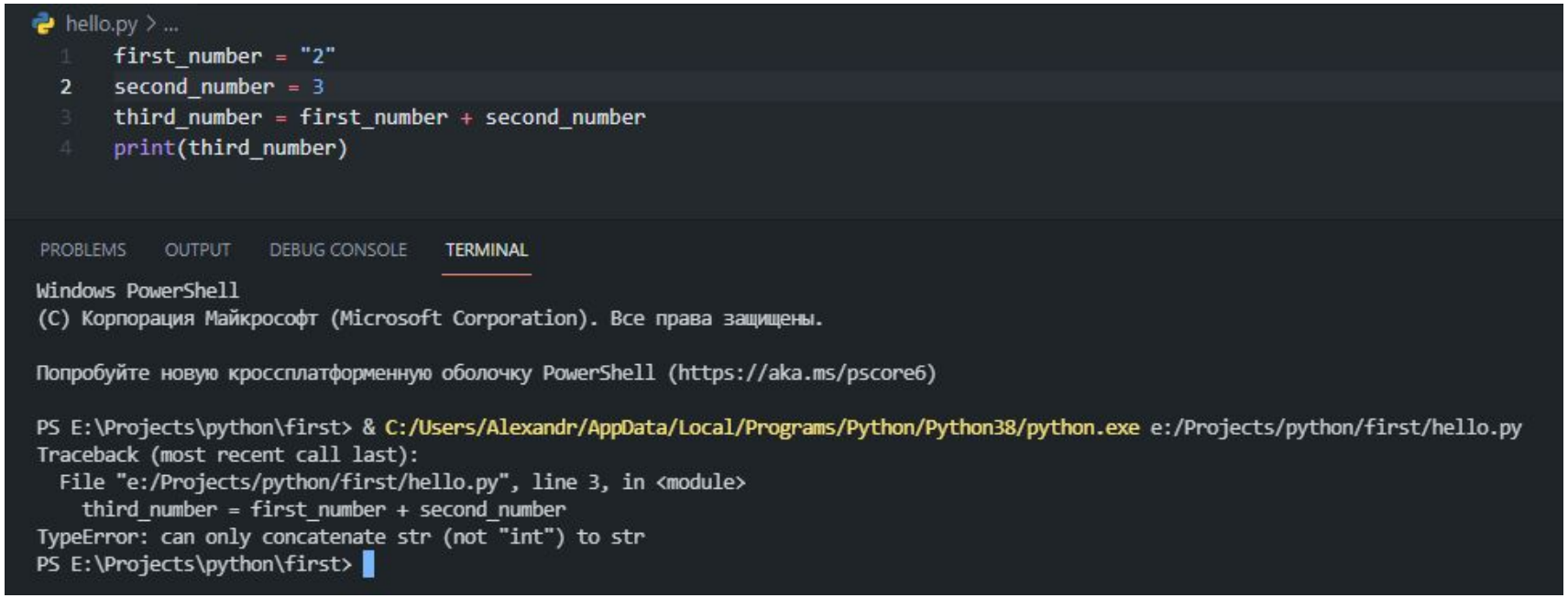
```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
15
12
48
PS E:\Projects\python\first>
```

Операции с числами

Функции преобразования чисел

Ряд встроенных функций в Python позволяют работать с числами. В частности, функции int() и float() позволяют привести значение к типу int и float соответственно.

Например, пусть у нас будет следующий код:



Мы ожидаем, что "2" + 3 будет равно 5. Однако этот код сгенерирует исключение, так как первое число на самом деле представляет строку.

Операции с числами

И чтобы все заработало как надо, необходимо привести строку к числу с помощью функции int():

hello.py > ...

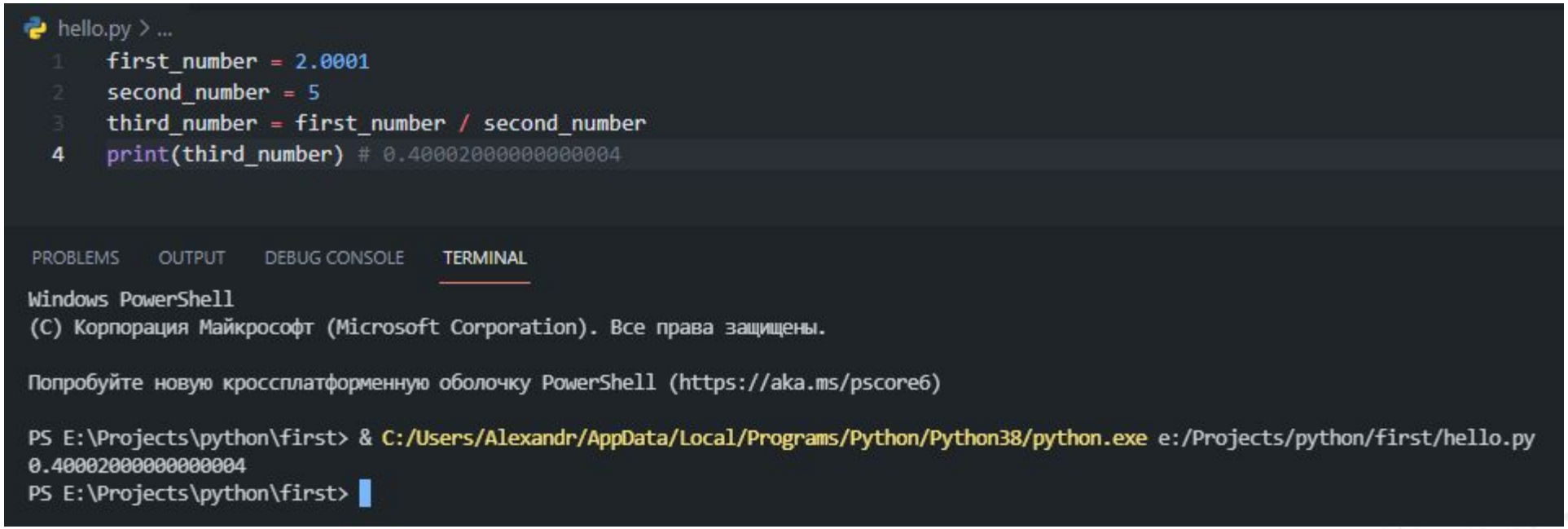
```
1   first_number = "2"
2   second_number = 3
3   third_number = int(first_number) + second_number
4   print(third_number) # 5
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
5
PS E:\Projects\python\first>
```

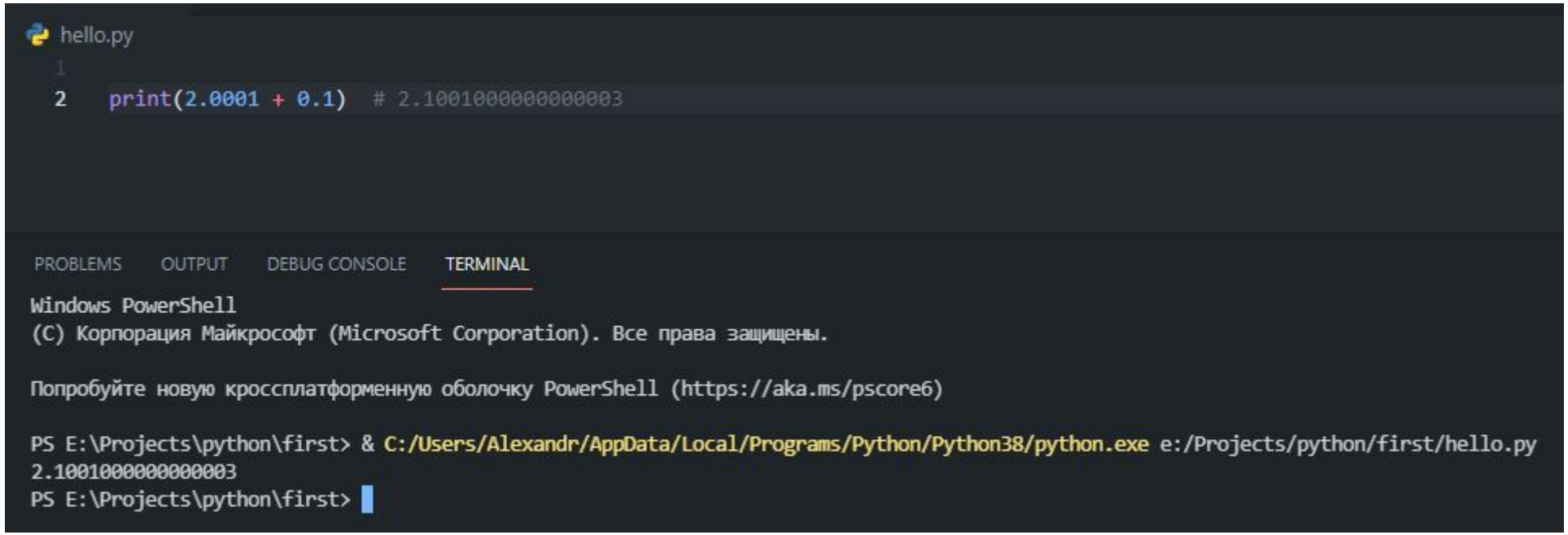
Операции с числами

Аналогичным образом действует функция float(), которая преобразует в число с плавающей точкой. Но вообще с дробными числами надо учитывать, что результат операций с ними может быть не совсем точным. Например:



Операции с числами

В данном случае мы ожидаем получить число 0.40002, однако в конце через ряд нулей появляется еще какая-то четверка. Или еще одно выражение:



Операции с числами

В этот случае для округления результата мы можем использовать функцию round():

hello.py > ...

```
1   first_number = 2.0001
2   second_number = 0.1
3   third_number = first_number + second_number
4   print(round(third_number, 4))  # 2.1001
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
2.1001
PS E:\Projects\python\first>
```

Первый параметр функции - округляемое число, а второй - сколько знаков после запятой должно содержать получаемое число.

Операции с числами

Представление числа

При обычном определении числовой переменной она получает значение в десятичной системе. Но кроме десятичной в Python мы можем использовать двоичную, восьмеричную и шестнадцатеричную системы.

Для определения числа в двоичной системе перед его значением ставится 0 и префикс **b**

Для определения числа в восьмеричной системе перед его значением ставится 0 и префикс **o**

Для определения числа в шестнадцатеричной системе перед его значением ставится 0 и префикс **x**

Операции с числами

Пример:

hello.py > ...

```python
x = 0b101   # 101 в двоичной системе равно 5
print(x)

a = 0o11    # 11 в восьмеричной системе равно 9
print(a)

y = 0x0a    # a в шестнадцатеричной системе равно 10
print(y)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
5
9
10
PS E:\Projects\python\first>
```

Операции с числами

И с числами в других системах измерения также можно проводить арифметические операции:

hello.py > ...

```
1   x = 0b101    # 5
2   y = 0x0a     # 10
3   z = x + y    # 15
4   print("{0} in binary {0:08b}   in hex {0:02x} in octal {0:02o}".format(z))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
15 in binary 00001111   in hex 0f in octal 17
PS E:\Projects\python\first>
```

Операции с числами

Для вывода числа в различных системах исчисления используются функция format, которая вызывается у строки. В эту строку передаются различные форматы. Для двоичной системы "{0:08b}", где число 8 указывает, сколько знаков должно быть в записи числа. Если знаков указано больше, чем требуется для числа, то ненужные позиции заполняются нулями. Для шестнадцатеричной системы применяется формат "{0:02x}". И здесь все аналогично - запись числа состоит из двух знаков, если один знак не нужен, то вместо него вставляется ноль. А для записи в восьмеричной системе исполуется формат "{0:02o}".

Условные выражения

Ряд операций представляют условные выражения. Все эти операции принимают два операнда и возвращают логическое значение, которое в Python представляет тип boolean. Существует только два логических значения - True (выражение истинно) и False (выражение ложно).

Условные выражения

Операции сравнения

Простейшие условные выражения представляют операции сравнения, которые сравнивают два значения. Python поддерживает следующие операции сравнения:

==

Возвращает True, если оба операнда равны. Иначе возвращает False.

!=

Возвращает True, если оба операнда НЕ равны. Иначе возвращает False.

> (больше чем)

Возвращает True, если первый операнд больше второго.

< (меньше чем)

Возвращает True, если первый операнд меньше второго.

>= (больше или равно)

Возвращает True, если первый операнд больше или равен второму.

<= (меньше или равно)

Возвращает True, если первый операнд меньше или равен второму.

Условные выражения

Примеры операций сравнения:

hello.py > ...

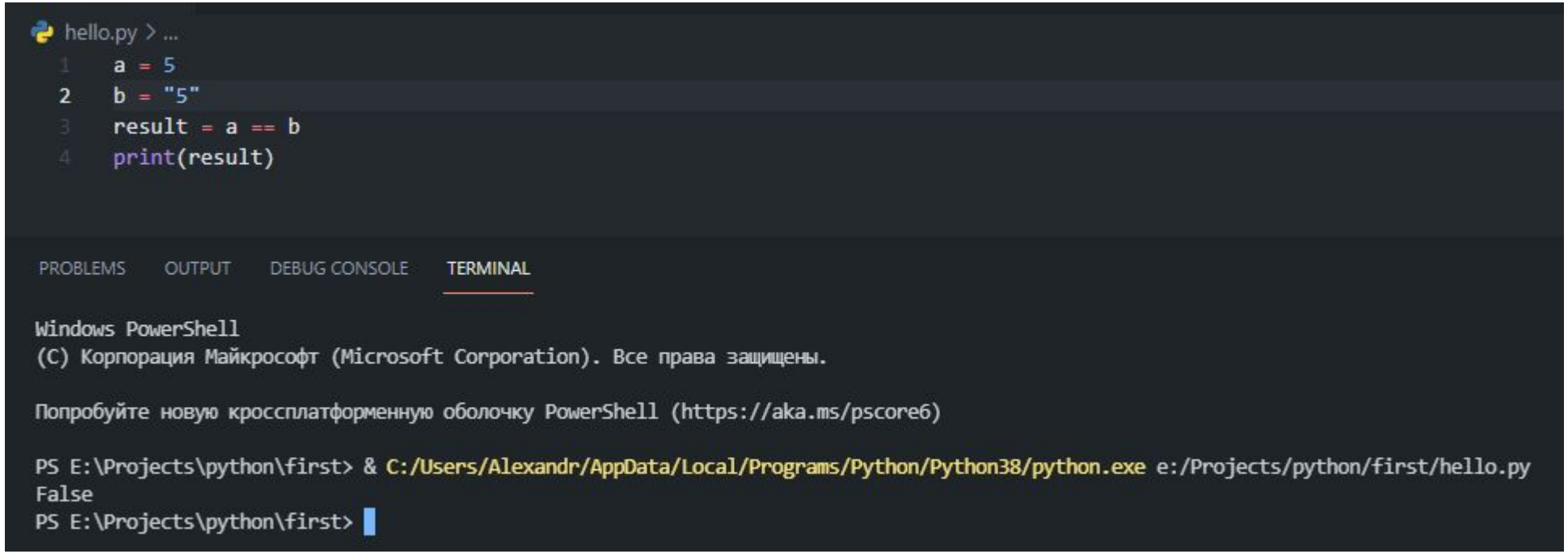
```python
a = 5
b = 6
result = 5 == 6  # сохраняем результат операции в переменную
print(result)  # False - 5 не равно 6
print(a != b)  # True
print(a > b)  # False - 5 меньше 6
print(a < b)  # True

bool1 = True
bool2 = False
print(bool1 == bool2)  # False - bool1 не равно bool2
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
False
True
False
True
False
PS E:\Projects\python\first>
```

Условные выражения

Операции сравнения могут сравнивать различные объекты - строки, числа, логические значения, однако оба операнда операции должны представлять один и тот же тип.



Условные выражения

Логические операции

Для создания составных условных выражений применяются логические операции. В Python имеются следующие логические операторы:

and (логическое умножение)

Возвращает True, если оба выражения равны True

hello.py > ...

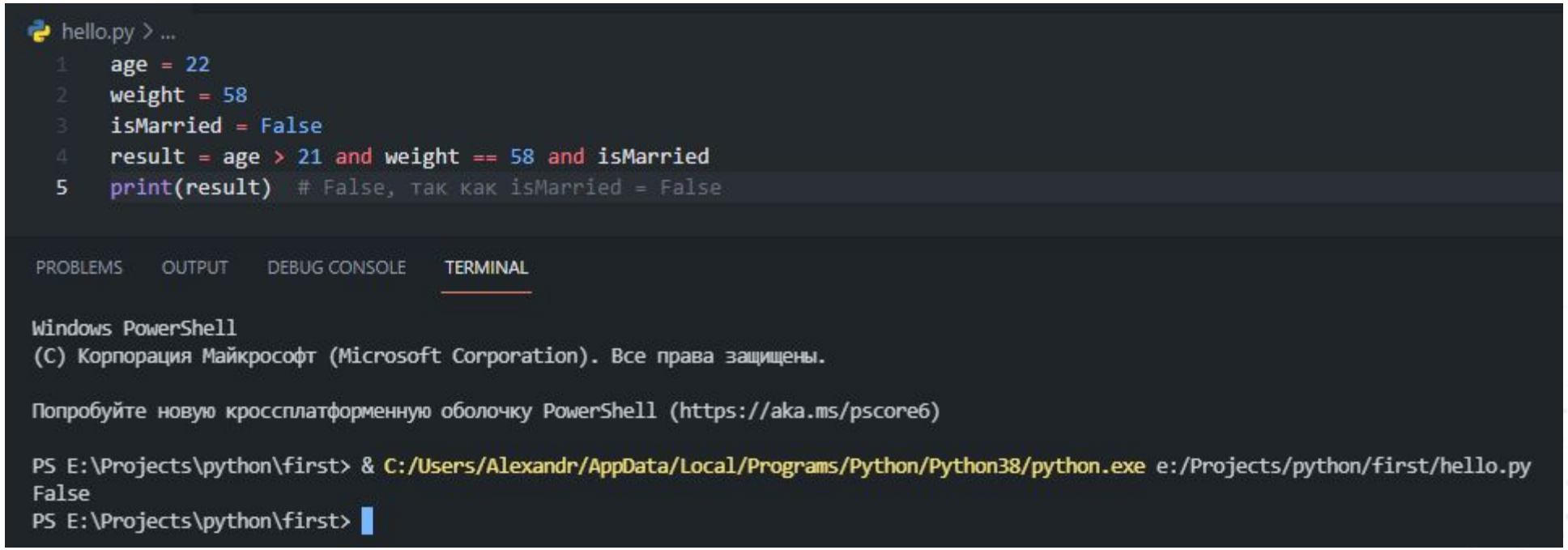
```
1   age = 22
2   weight = 58
3   result = age > 21 and weight == 58
4   print(result)  # True
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
True
PS E:\Projects\python\first>
```

Условные выражения

В данном случае оператор and сравнивает результаты двух выражений: age > 21 weight == 58. И если оба этих выражений возвращают True, то оператор and также возвращает True. Причем в качестве одно из выражений необязательно выступает операция сравнения: это может быть другая логическая операция или просто переменная типа boolean, которая хранит True или False.



Условные выражения

or (логическое сложение)

Возвращает True, если хотя бы одно из выражений равно True

hello.py > ...

```
1   age = 22
2   isMarried = False
3   result = age > 21 or isMarried
4   print(result)  # True, так как выражение age > 21 равно True
```

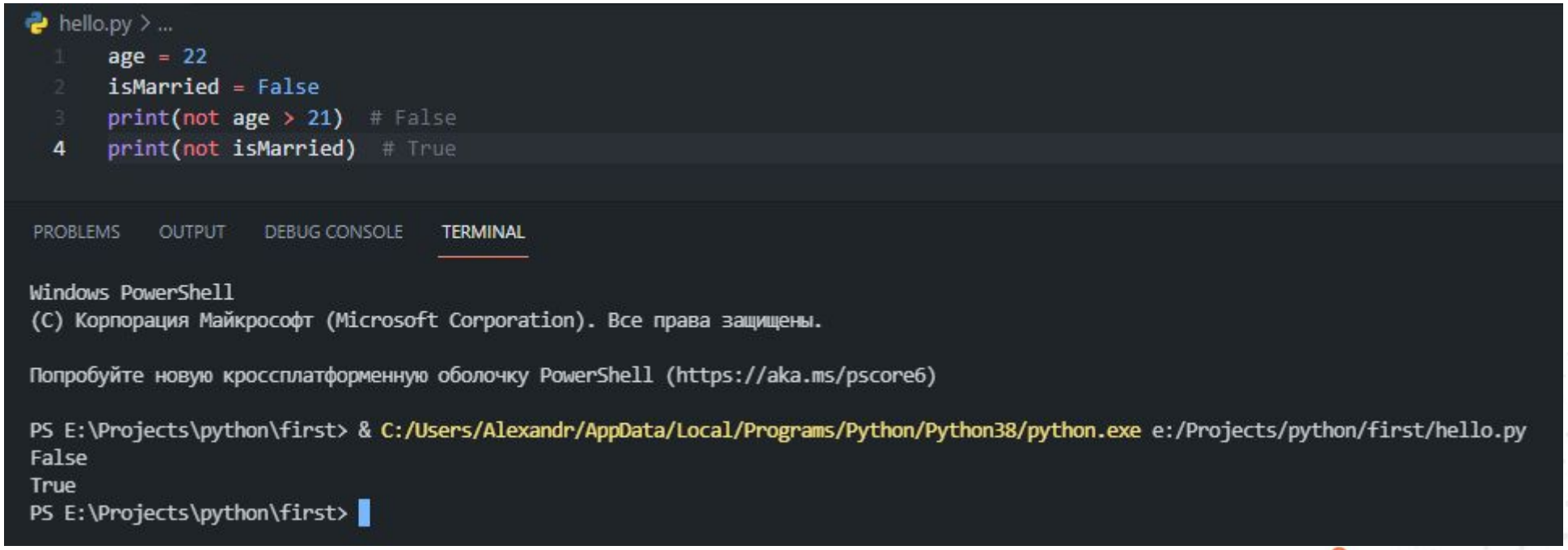
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
True
PS E:\Projects\python\first>
```

Условные выражения

not (логическое отрицание)

Возвращает True, если выражение равно False



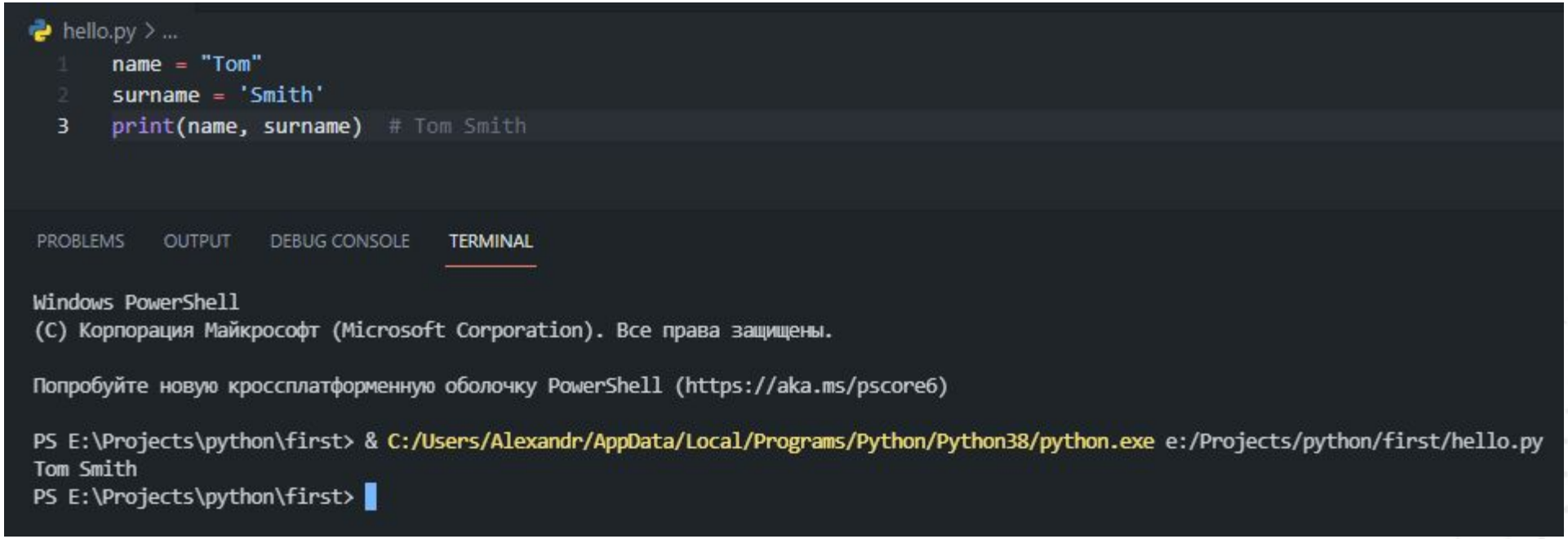
Условные выражения

Если один из операндов оператора **and** возвращает False, то другой операнд уже не оценивается, так как оператор в любом случае возвратит False. Подобное поведение позволяет немного увеличить производительность, так как не приходится тратить ресурсы на оценку второго операнда.

Аналогично если один из операндов оператора **or** возвращает True, то второй операнд не оценивается, так как оператор в любом случае возвратит True.

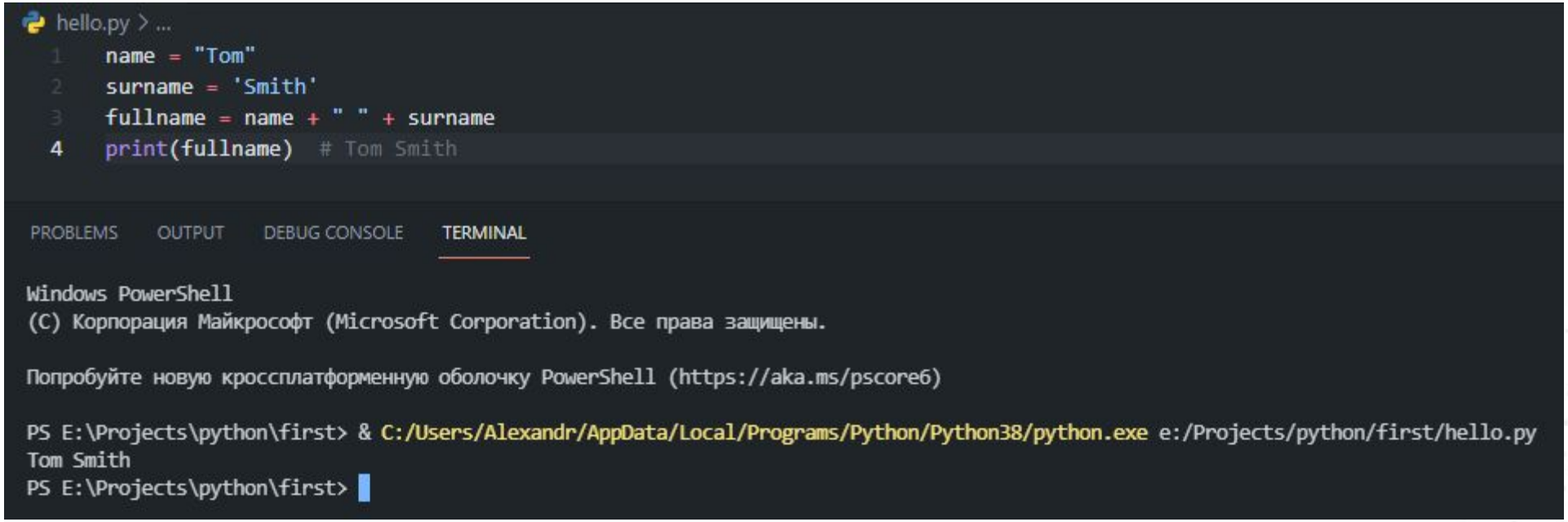
Операции со строками

Строка представляет последовательность символов в кодировке Unicode, заключенных в кавычки. Причем в Python мы можем использовать как одинарные, так и двойные кавычки:



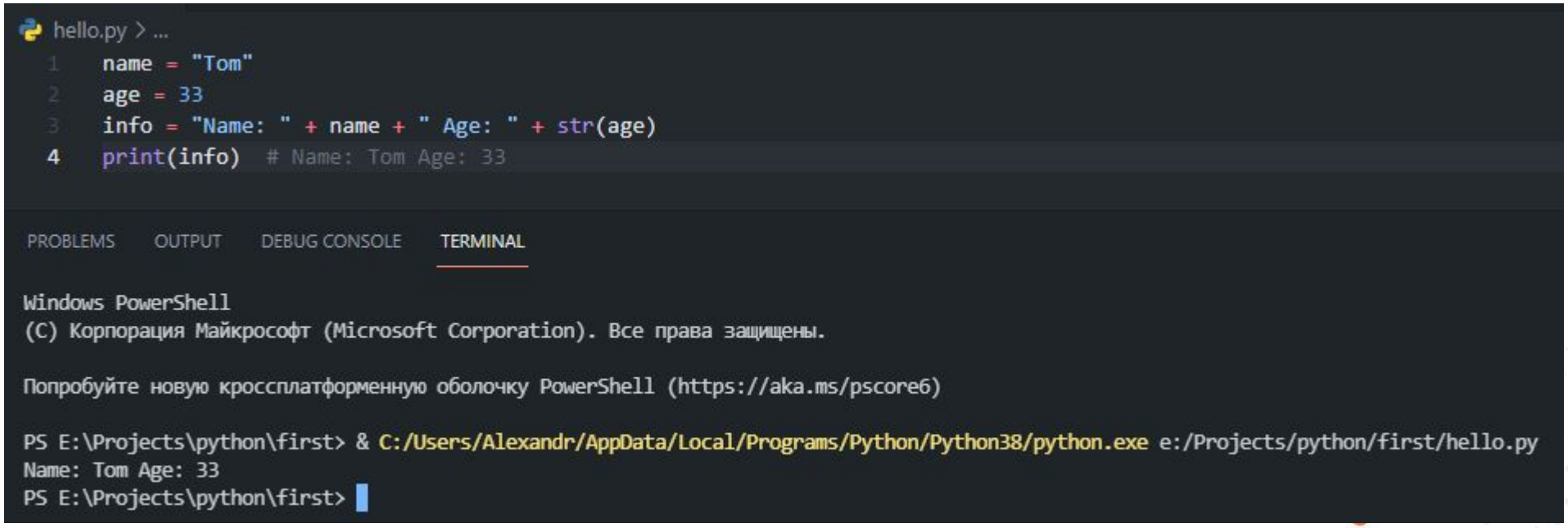
Операции со строками

Одной из самых распространенных операций со строками является их объединение или конкатенация. Для объединения строк применяется знак плюса:



Операции со строками

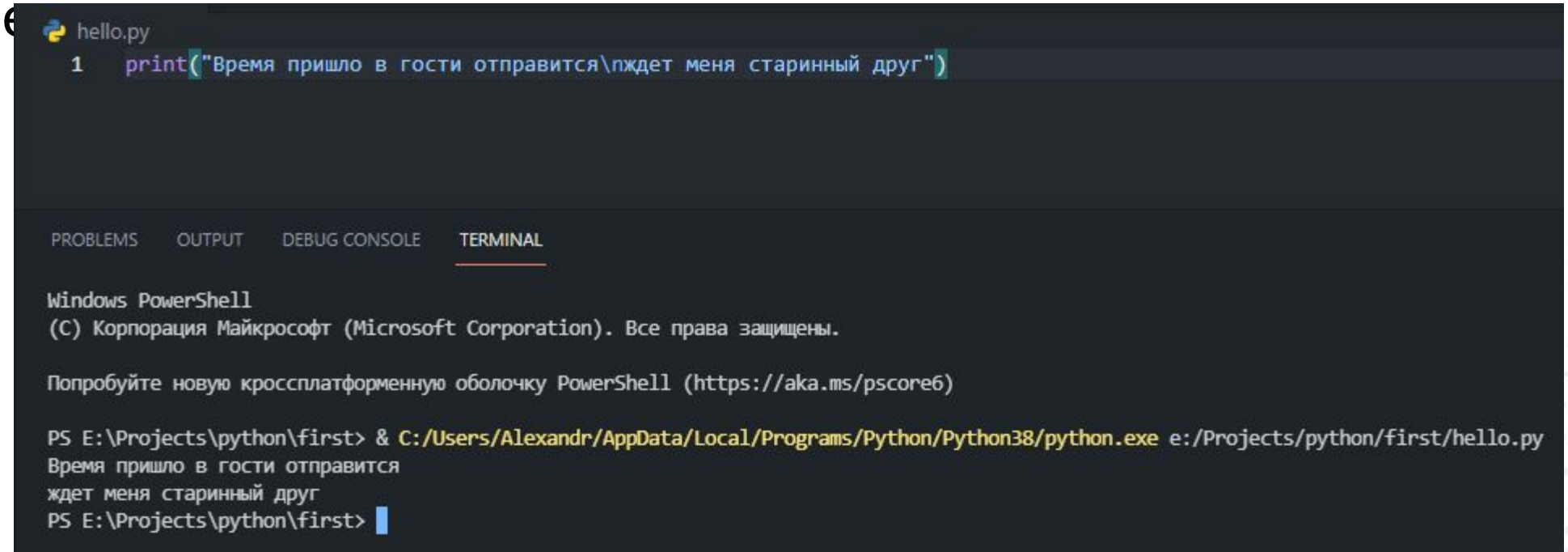
С объединением двух строк все просто, но что, если нам надо сложить строку и число? В этом случае необходимо привести число к строке с помощью функции str():



Операции со строками

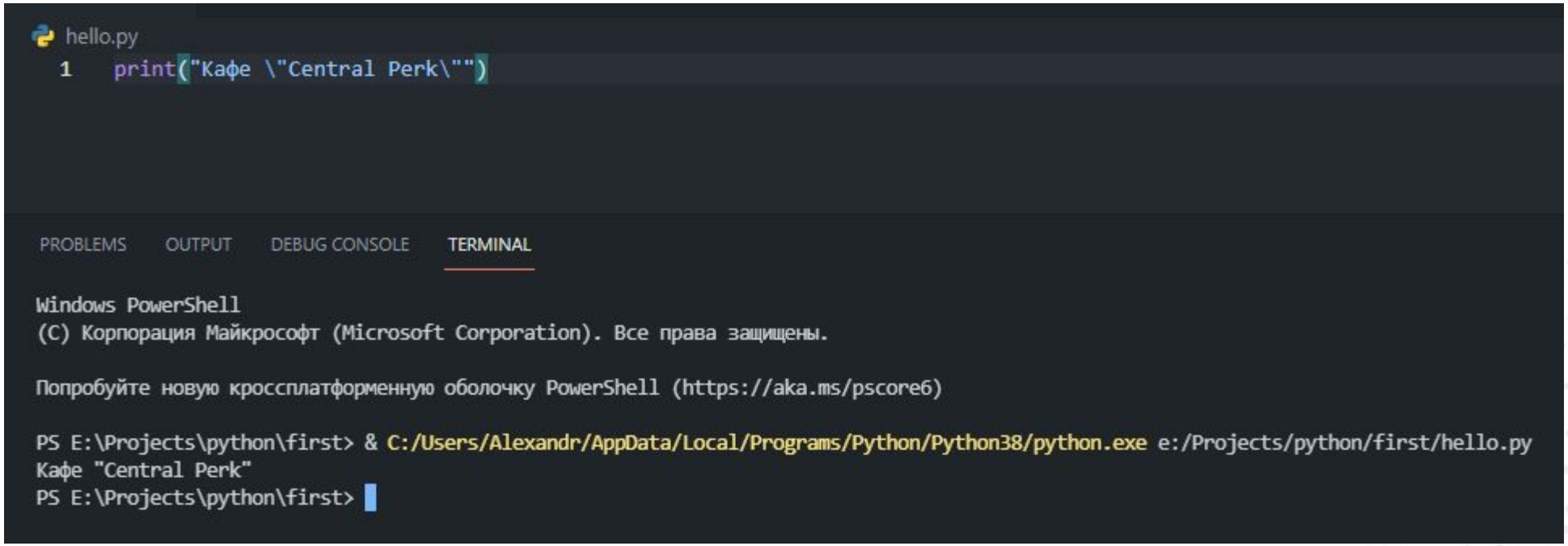
Эскейп-последовательности

Кроме стандартных символов строки могут включать управляющие эскейп-последовательности, которые интерпретируются особым образом. Например, последовательность \n представляет перевод строки. Поэтому сле



Операции со строками

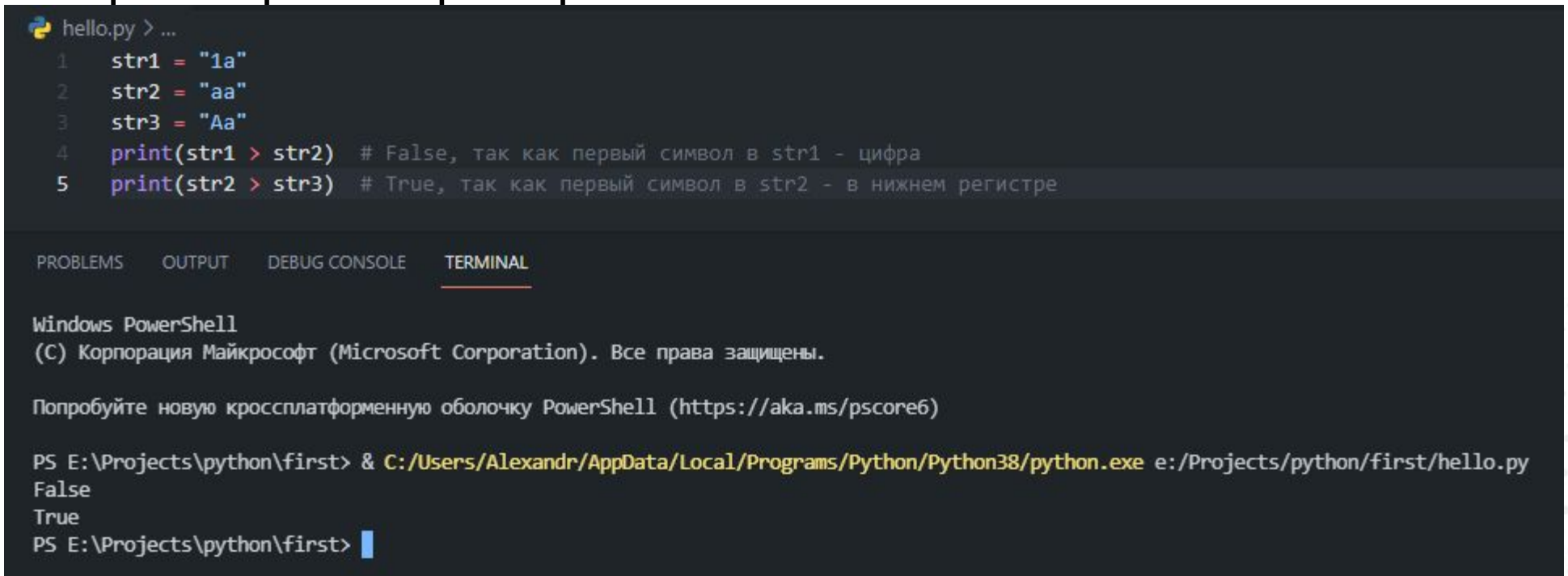
Кроме того, существуют символы, которые вроде бы сложно использовать в строке. Например, кавычки. И чтобы отобразить кавычки (как двойные, так и одинарные) внутри строки, перед ними ставится слеш:



Операции со строками

Сравнение строк

Особо следует сказать о сравнении строк. При сравнении строк принимается во внимание символы и их регистр. Так, цифровой символ условно меньше, чем любой алфавитный символ. Алфавитный символ в верхнем регистре условно меньше, чем алфавитные символы в нижнем регистре. Например:



Операции со строками

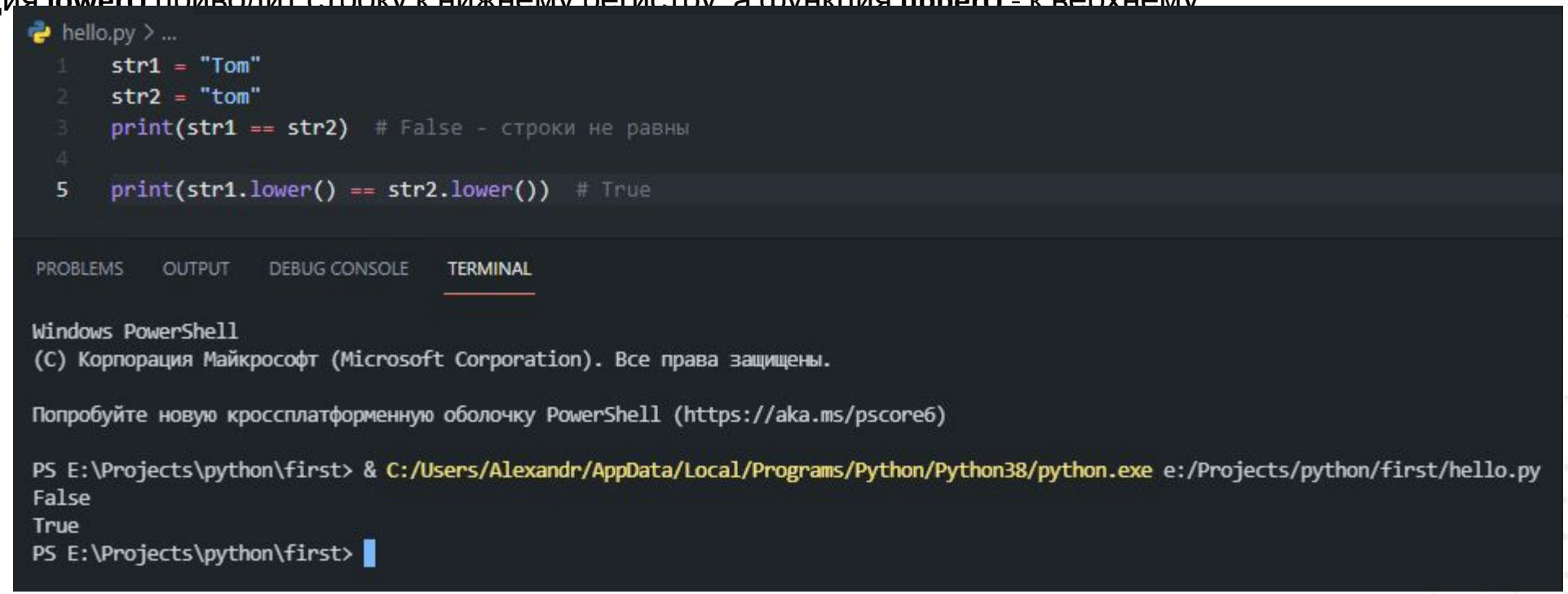
Поэтому строка "1a" условно меньше, чем строка "aa". Вначале сравнение идет по первому символу. Если начальные символы обоих строк представляют цифры, то меньшей считается меньшая цифра, например, "1a" меньше, чем "2a".

Если начальные символы представляют алфавитные символы в одном и том же регистре, то смотрят по алфавиту. Так, "aa" меньше, чем "ba", а "ba" меньше, чем "ca".

Если первые символы одинаковые, в расчет берутся вторые символы при их наличии.

Зависимость от регистра не всегда желательна, так как по сути мы имеем дело с одинаковыми строками. В этом случае перед сравнением мы можем привести обе строки к одному из регистров.

Функция **lower()** приводит строку к нижнему регистру, а функция **upper()** - к верхнему



Условная конструкция if

Условные конструкции используют условные выражения и в зависимости от их значения направляют выполнение программы по одному из путей. Одна из таких конструкций - это конструкция **if**. Она имеет следующее формальное определение:

if логическое_выражение:

    инструкции

[elif логическое выражение:

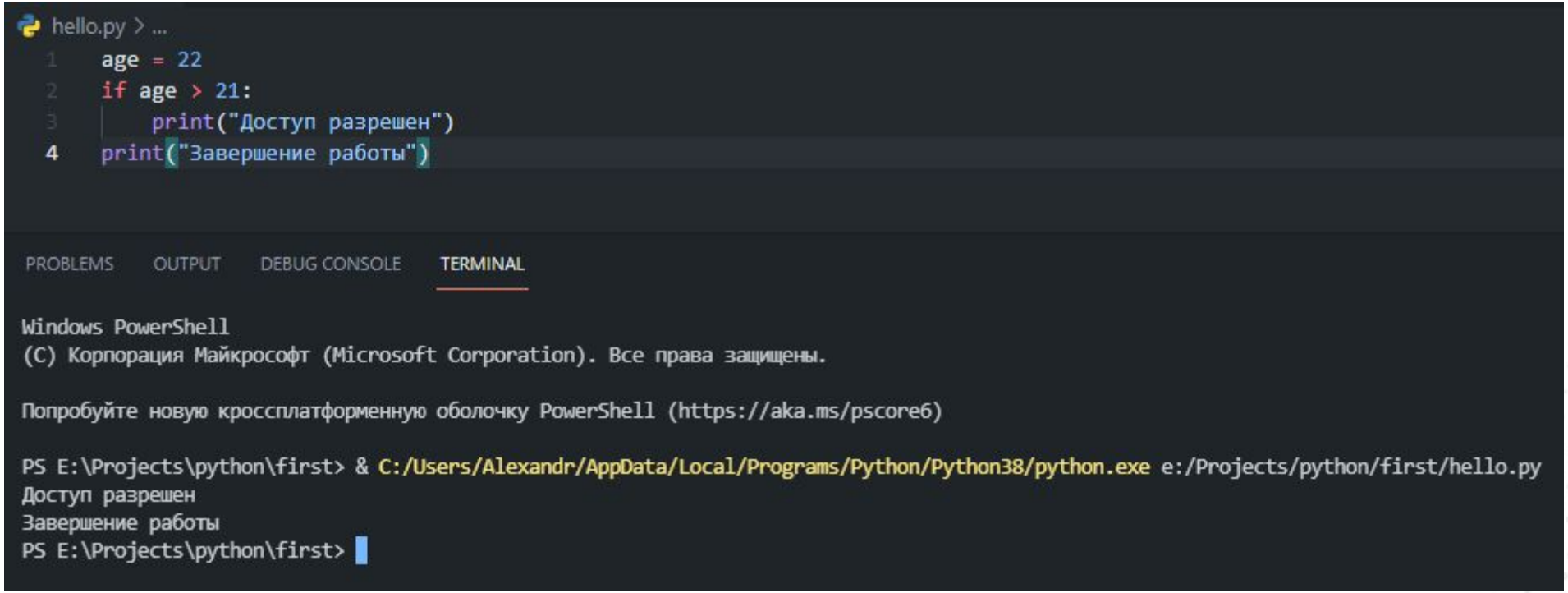
    инструкции]

[else:

    инструкции]

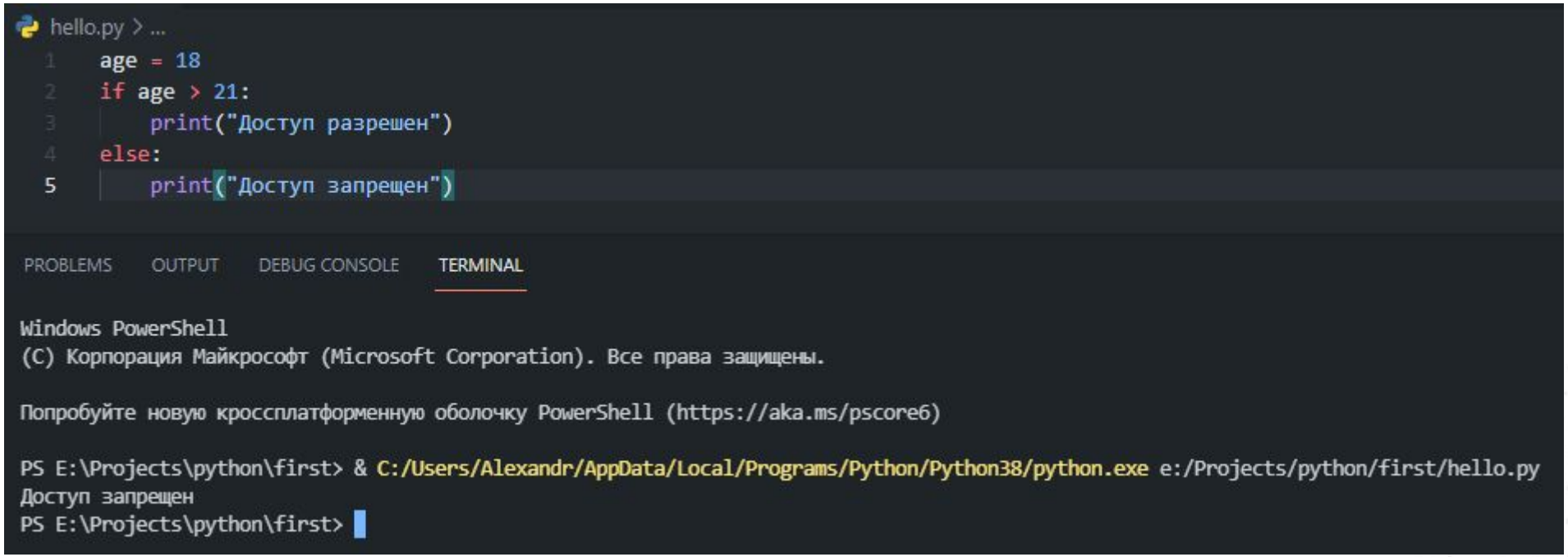
Условная конструкция if

В самом простом виде после ключевого слова if идет логическое выражение. И если это логическое выражение возвращает True, то выполняется последующий блок инструкций, каждая из которых должна начинаться с новой стоки и должна иметь отступы от начала строки:



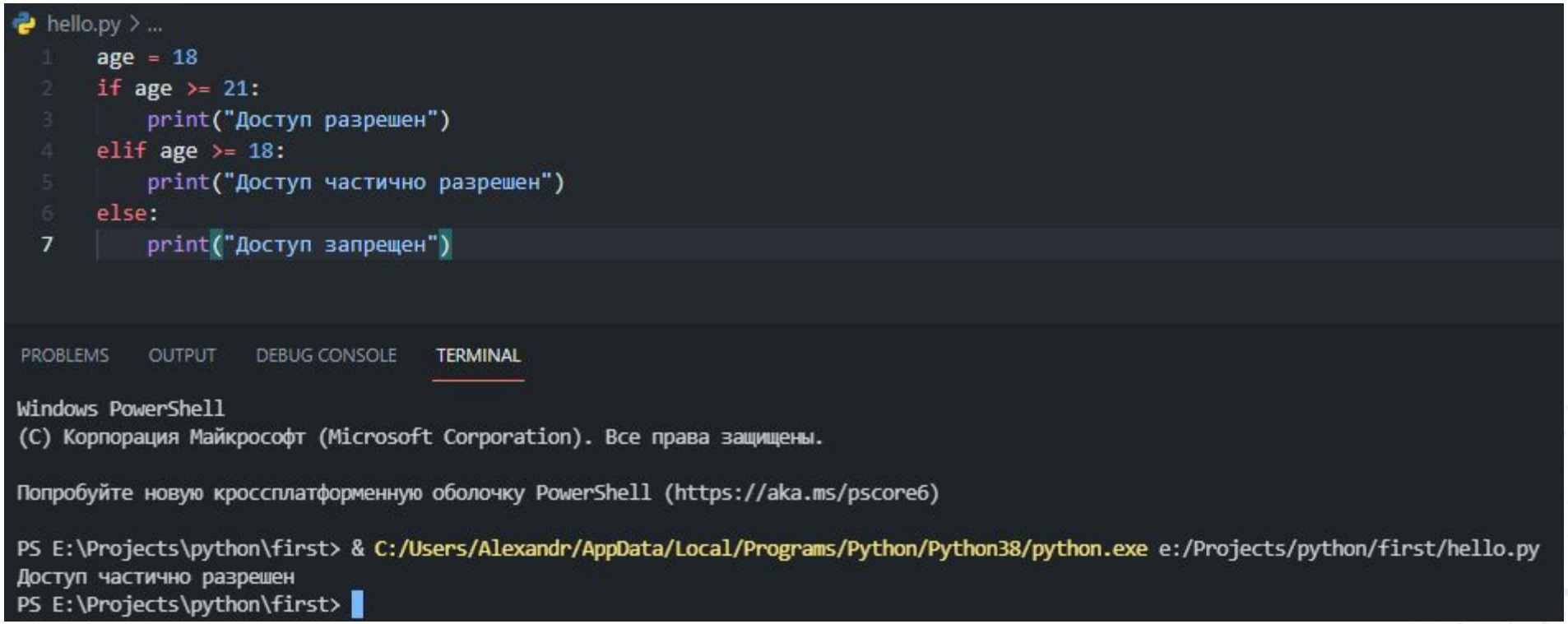
Условная конструкция if

Если вдруг нам надо определить альтернативное решение на тот случай, если условное выражение возвратит False, то мы можем использовать блок **else**:



Условная конструкция if

Если необходимо ввести несколько альтернативных условий, то можно использовать дополнительные блоки **elif**, после которого идет блок инструкций.



Условная конструкция if

Вложенные конструкции if

Конструкция if в свою очередь сама может иметь вложенные конструкции if:

hello.py > ...

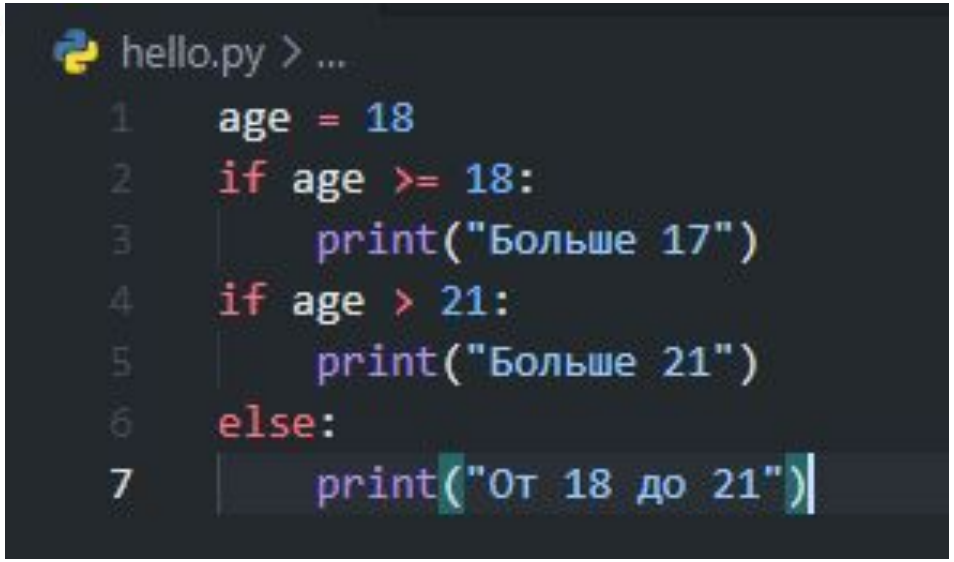
```
1   age = 18
2   if age >= 18:
3       print("Больше 17")
4       if age > 21:
5           print("Больше 21")
6       else:
7           print("От 18 до 21")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Больше 17
От 18 до 21
PS E:\Projects\python\first>
```

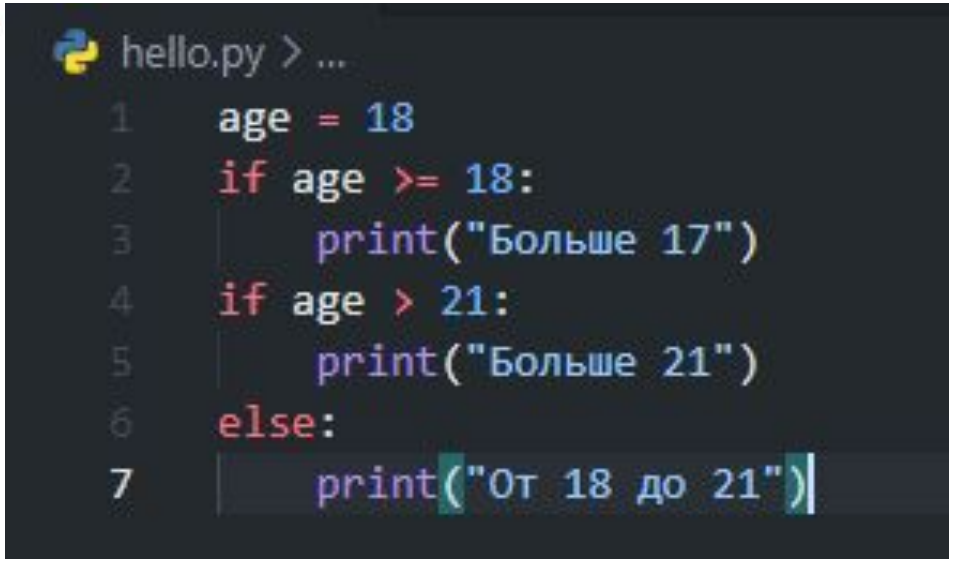
Условная конструкция if

Стоит учитывать, что вложенные выражения if также должны начинаться с отступов, а инструкции во вложенных конструкциях также должны иметь отступы. Отступы, расставленные не должным образом, могут изменить логику программы. Так, предыдущий пример НЕ аналогичен следующему:



Условная конструкция if

Стоит учитывать, что вложенные выражения if также должны начинаться с отступов, а инструкции во вложенных конструкциях также должны иметь отступы. Отступы, расставленные не должным образом, могут изменить логику программы. Так, предыдущий пример НЕ аналогичен следующему:



Циклы

Циклы позволяют повторять некоторое действие в зависимости от соблюдения некоторого условия.

Цикл while

Первый цикл, который мы рассмотрим, это цикл **while**. Он имеет следующее формальное определение:

```
while условное_выражение:
    инструкции
```

После ключевого слова while указывается условное выражение, и пока это выражение возвращает значение True, будет выполняться блок инструкций, который идет далее.

Все инструкции, которые относятся к циклу while, располагаются на последующих строках и должны иметь отступ от начала строки.

Циклы

hello.py > ...

```python
choice = "y"

while choice.lower() == "y":
    print("Привет")
    choice = input("Для продолжения нажмите Y, а для выхода любую другую клавишу: ")
print("Работа программы завешена")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Привет
Для продолжения нажмите Y, а для выхода любую другую клавишу: y
Привет
Для продолжения нажмите Y, а для выхода любую другую клавишу: h
Работа программы завешена
PS E:\Projects\python\first>
```

Циклы

В данном случае цикл while будет продолжаться, пока переменная choice содержит латинскую букву "Y" или "y".

Сам блок цикла состоит из двух инструкций. Сначала выводится сообщение "Привет", а потом вводится новое значение для переменной choice. И если пользователь нажмет какую-то другую клавишу, отличную от Y, произойдет выход из цикла, так как условие choice.lower() == "y" вернет значение False. Каждый такой проход цикла называется итерацией.

Также обратите внимание, что последняя инструкция print("Работа программы завешена") не имеет отступов от начала строки, поэтому она не входит в цикл while.

Циклы

Другой пример - вычисление факториала:

hello.py > ...

```python
#! Программа по вычислению факториала

number = int(input("Введите число: "))
i = 1
factorial = 1
while i <= number:
    factorial *= i
    i += 1
print("Факториал числа", number, "равен", factorial)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Введите число: 6
Факториал числа 6 равен 720
PS E:\Projects\python\first>
```

Циклы

Цикл for

Другой тип циклов представляет конструкция **for**. Цикл for вызывается для каждого числа в некоторой коллекции чисел. Коллекция чисел создается с помощью функции **range()**. Формальное определение цикла for:

```
for int_var in функция_range:
    инструкции
```

После ключевого слова **for** идет переменная int_var, которая хранит целые числа (название переменной может быть любое), затем ключевое слово **in**, вызов функции range() и двоеточие.

А со следующей строки располагается блок инструкций цикла, которые также должны иметь отступы от начала строки.

Циклы

При выполнении цикла Python последовательно получает все числа из коллекции, которая создается функцией range, и сохраняет эти числа в переменной int_var. При первом проходе цикл получает первое число из коллекции, при втором - второе число и так далее, пока не переберет все числа. Когда все числа в коллекции будут перебраны, цикл завершает свою работу.

Циклы

Рассмотрим на примере вычисления факториала:

hello.py > ...

```python
#! Программа по вычислению факториала

number = int(input("Введите число: "))
factorial = 1
for i in range(1, number+1):
    factorial *= i
print("Факториал числа", number, "равен", factorial)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Введите число: 6
Факториал числа 6 равен 720
PS E:\Projects\python\first>
```

Циклы

Вначале вводим с консоли число. В цикле определяем переменную i, в которую сохраняются числа из коллекции, создаваемой функцией range.

Функция range здесь принимает два аргумента - начальное число коллекции (здесь число 1) и число, до которого надо добавлять числа (то есть number +1).

Допустим, с консоли вводится число 6, то вызов функции range приобретает следующую форму:

range(1, 6+1)

Эта функция будет создавать коллекцию, которая будет начинаться с 1 и будет последовательно наполняться целыми числами вплоть до 7. То есть это будет коллекция [1, 2, 3, 4, 5, 6].

При выполнении цикла из этой коллекции последовательно будут передаваться числа в переменную i, а в самом цикле будет происходить умножение переменной i на переменную factorial. В итоге мы получим факториал числа.

Консольный вывод программы:

Циклы

Функция range

Функция range имеет следующие формы:

range(stop): возвращает все целые числа от 0 до stop

range(start, stop): возвращает все целые числа в промежутке от start (включая) до stop (не включая). Выше в программе факториала использована именно эта форма.

range(start, stop, step): возвращает целые числа в промежутке от start (включая) до stop (не включая), которые увеличиваются на значение step

Циклы

Пример

hello.py > ...

```python
# Примеры вызовов функции range
range(5)            # 0, 1, 2, 3, 4
range(1, 5)         # 1, 2, 3, 4
range(2, 10, 2)     # 2, 4, 6, 8
range(5, 0, -1)     # 5, 4, 3, 2, 1

# Например, выведем последовательно все числа от 0 до 4:
for i in range(5):
    print(i, end=" ")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
0 1 2 3 4
PS E:\Projects\python\first>
```

Циклы

Вложенные циклы

Одни циклы внутри себя могут содержать другие циклы. Рассмотрим на примере вывода таблицы умножения.

hello.py > ...

```
for i in range(1, 10):
    for j in range(1, 10):
        print(i * j, end="\t")
    print("\n")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.
1       2       3       4       5       6       7       8       9

2       4       6       8       10      12      14      16      18

3       6       9       12      15      18      21      24      27

4       8       12      16      20      24      28      32      36

5       10      15      20      25      30      35      40      45

6       12      18      24      30      36      42      48      54

7       14      21      28      35      42      49      56      63

8       16      24      32      40      48      56      64      72

9       18      27      36      45      54      63      72      81

PS E:\Projects\python\first>
```

Циклы

Внешний цикл for i in range(1, 10) срабатывает 9 раз, так как в коллекции, возвращаемой функцией range, 9 чисел.
Внутренний цикл for j in range(1, 10) срабатывает 9 раз для одной итерации внешнего цикла, и соответственно 81 раз для всех итераций внешнего цикла.

Циклы

Выход из цикла. break и continue

Для управления циклом мы можем использовать специальные операторы **break** и **continue**. Оператор break осуществляет выход из цикла. А оператор continue выполняет переход к следующей итерации цикла.

Оператор break может использоваться, если в цикле образуются условия, которые несовместимы с его дальнейшим выполнением.

Циклы

Пример

hello.py > ...

```
1   for i in 'hello world':
2       if i == 'a':
3           break
4   else:
5       print('Буквы a в строке нет')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Буквы a в строке нет
PS E:\Projects\python\first>
```

Циклы

Пример 2

hello.py > ...

```python
index = 0

while True:
    if index > 5:
        break
    else:
        index = index + 1
    print(index)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
1
2
3
4
5
6
PS E:\Projects\python\first>
```

Функции

Функции представляют блок кода, который выполняет определенную задачу и который можно повторно использовать в других частях программы. Формальное определение функции:

```
def имя_функции ([параметры]):
    инструкции
```

Определение функции начинается с выражения **def**, которое состоит из имени функции, набора скобок с параметрами и двоеточия. Параметры в скобках необязательны. А со следующей строки идет блок инструкций, которые выполняет функция. Все инструкции функции имеют отступы от начала строки.

Функции

Пример

hello.py > ...

```python
# функция без параметров
def say_hello():
    print("Hello")

# функция с параметрами
def say_hello_to(name):
    print("Hello,",name)

# вызов функций
say_hello()
say_hello_to("Alex")
```

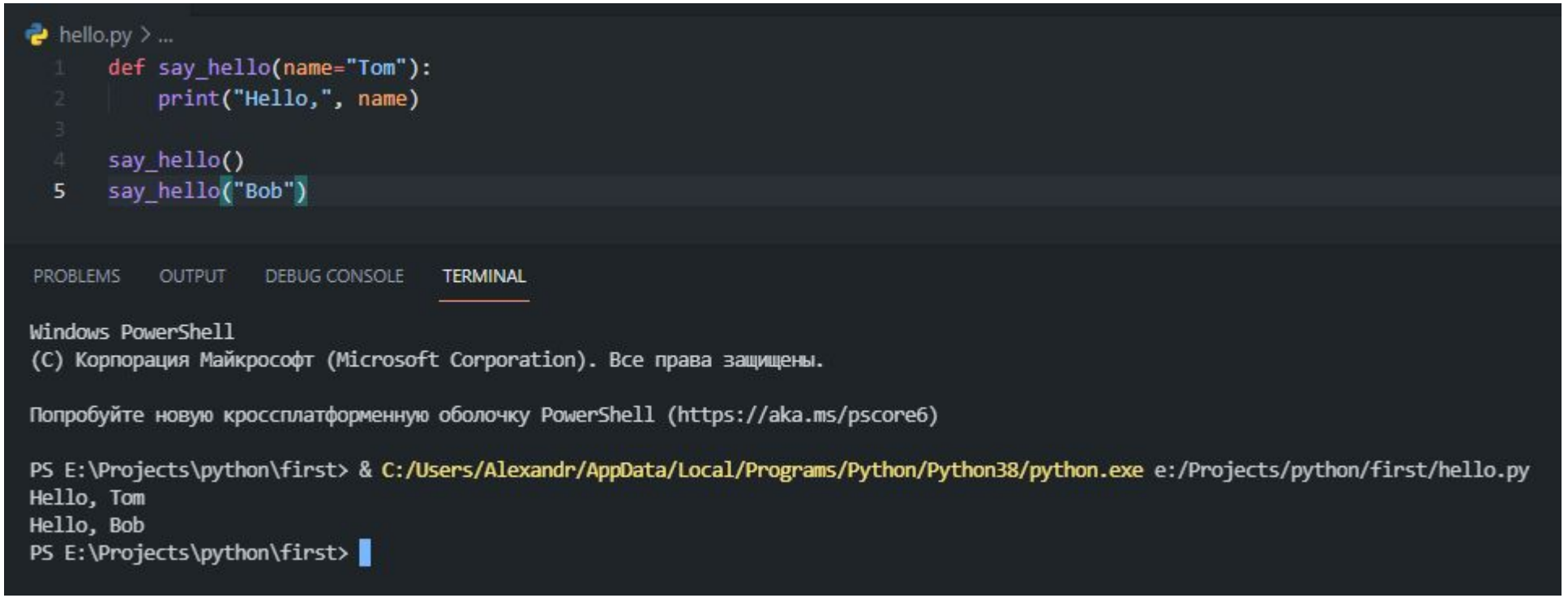
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Hello
Hello, Alex
PS E:\Projects\python\first>
```

Функции

Значения по умолчанию

Некоторые параметры функции мы можем сделать необязательными, указав для них значения по умолчанию при определении функции. Например:



Здесь параметр name является необязательным. И если мы не передаем при вызове функции для него значение, то применяется значение по умолчанию, то есть строка "Tom".

Функции

Именованные параметры

При передаче значений функция сопоставляет их с параметрами в том порядке, в котором они передаются. Например, пусть есть следующая функция:

hello.py > ...

```
1   def display_info(name, age):
2       print("Name:", name, "\t", "Age:", age)
3
4   display_info("Tom", 22)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Name: Tom       Age: 22
PS E:\Projects\python\first>
```

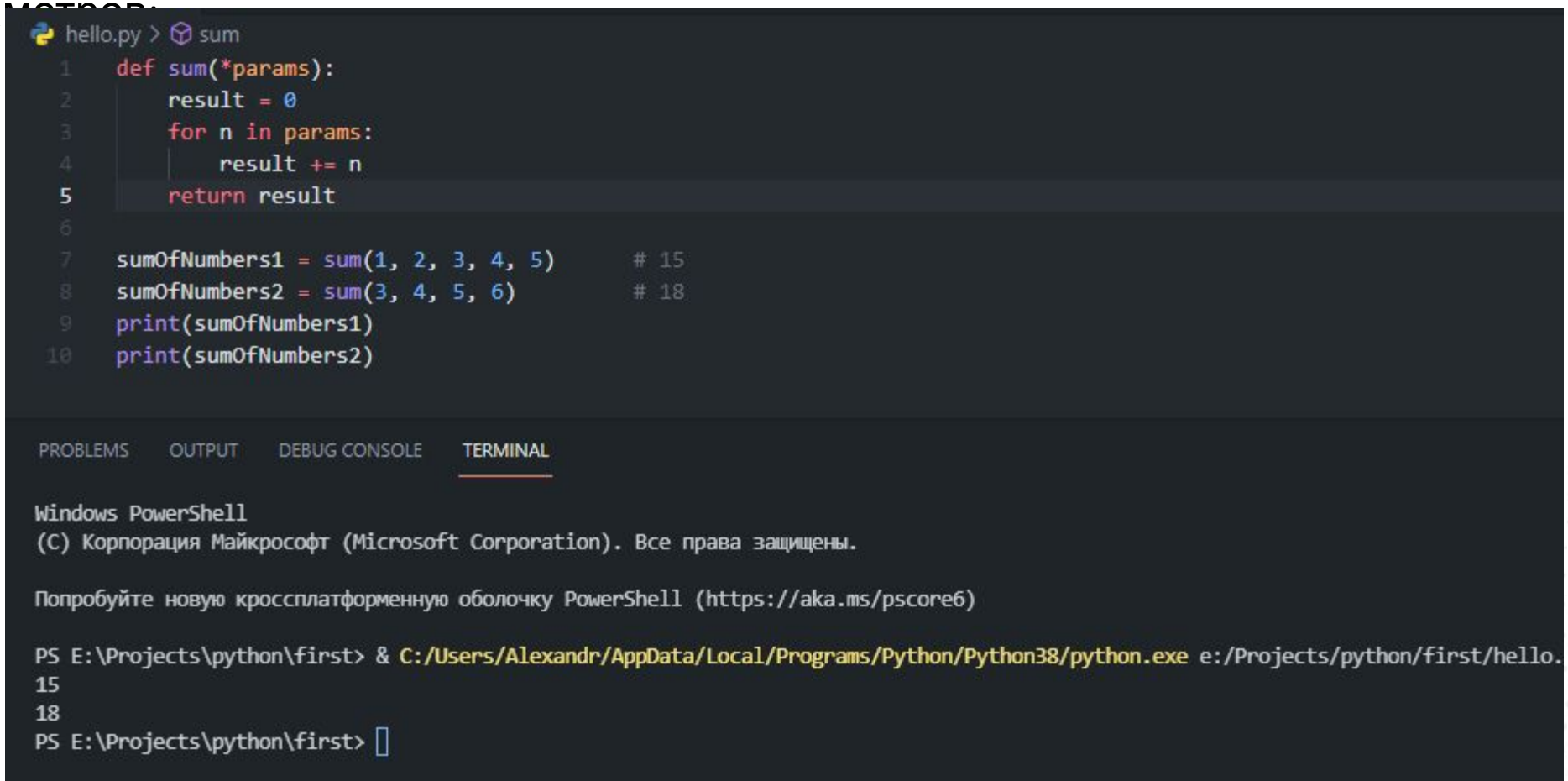
Функции

При вызове функции первое значение "Tom" передается первому параметру - параметру name, второе значение - число 22 передается второму параметру - age. И так далее по порядку. Использование именованных параметров позволяет переопределить порядок передачи:

Функции

Неопределенное количество параметров

С помощью символа звездочки можно определить неопределенное количество параметров:



Функции

В данном случае функция sum принимает один параметр - *params, но звездочка перед названием параметра указывает, что фактически на место этого параметра мы можем передать неопределенное количество значений или набор значений. В самой функции с помощью цикла for можно пройтись по этому набору и произвести с переданными значениями различные действия. Например, в данном случае возвращается сумма чисел.

Функции

Возвращение результата

Функция может возвращать результат. Для этого в функции используется оператор return, после которого указывается возвращаемое значение:

hello.py > ...

```
1   def exchange(usd_rate, money):
2       result = round(money/usd_rate, 2)
3       return result
4
5   result1 = exchange(60, 30000)
6   print(result1)
7   result2 = exchange(56, 30000)
8   print(result2)
9   result3 = exchange(65, 30000)
10  print(result3)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
500.0
535.71
461.54
PS E:\Projects\python\first>
```

Функции

В Python функция может возвращать сразу несколько значений:

hello.py > ...

```python
def create_default_user():
    name = "Tom"
    age = 33
    return name, age

user_name, user_age = create_default_user()
print("Name:", user_name, "\t Age:", user_age)
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Name: Tom        Age: 33
PS E:\Projects\python\first>
```

Здесь функция create_default_user возвращает два значения: name и age. При вызове функции эти значения по порядку присваиваются переменным user_name и user_age, и мы их можем использовать.

Функции

Функция main

В программе может быть определено множество функций. И чтобы всех их упорядочить, хорошей практикой считается добавление специальной функции main, в которой потом уже вызываются другие функции.

Функции

hello.py > ...

```python
def main():
    say_hello("Tom")
    usd_rate = 56
    money = 30000
    result = exchange(usd_rate, money)
    print("К выдаче", result, "долларов")


def say_hello(name):
    print("Hello,", name)


def exchange(usd_rate, money):
    result = round(money/usd_rate, 2)
    return result

# Вызов функции main
main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
(C) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

Попробуйте новую кроссплатформенную оболочку PowerShell (https://aka.ms/pscore6)

PS E:\Projects\python\first> & C:/Users/Alexandr/AppData/Local/Programs/Python/Python38/python.exe e:/Projects/python/first/hello.py
Hello, Tom
К выдаче 535.71 долларов
PS E:\Projects\python\first>
```